EIN ZIELSYSTEMIDENTISCHER ANSATZ FÜR DAS DOMÄNENSPEZIFISCHE RAPID PROTOTYPING IN DER INFORMATIONS- UND ELEKTROTECHNIK

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

vorgelegt der Fakultät für Maschinenbau
der Technischen Universität Ilmenau

von
Dipl.-Ing. (FH) Markus Kasper

Gutachter: Univ.-Prof. Dr.-Ing. habil. M. Weiß, Technische Universität Ilmenau
Univ.-Prof. Dr.-Ing. Prof. h. c. T. Bertram, Technische Universität Ilmenau
Prof. Dr.-Ing. H. Töpfer, Fachhochschule Esslingen

vorgelegt am 16. Juni 2004

wissenschaftliche Aussprache am 30. Mai 2005

urn:nbn:de:gbv:ilm1-2005000091


ETAS
STRAID System II
ES 1000
SERIAL
MCD
MCI
MSF
SCD
USER I/O
1 2 3
BAT
BUS
ON
Auto
6...34V
ES1130.2
ES1640
Probe Int.
ES1000.2

DANKSAGUNGEN / WIDMUNGEN

Die vorliegende Dissertation entstand im Rahmen einer Projektkooperation zwischen der ETAS GmbH und der Forschung und Vorausentwicklung der ROBERT BOSCH GmbH.

Mein besonderer Dank gilt Herrn Univ.-Prof. Dr.-Ing. habil. Mathias Weiß*, Leiter des Fachgebiets „Rechneranwendung im Maschinenbau“ der Technischen Universität Ilmenau* für die engagierte Betreuung meiner Arbeit sowie für die anregenden und hilfreichen Diskussionen.

Herrn Univ.-Prof. Dr.-Ing. Torsten Bertram
Leiter des Fachgebiets „Mechatronik“ der Technischen Universität Ilmenau,

Herrn Univ.-Prof. Dr.-Ing. habil. Eberhard Kallenbach
Technische Universität Ilmenau, sowie

Herrn Prof. Dr.-Ing. Harald Töpfer
Fachhochschule für Technik Esslingen,

danke ich für die wertvollen Anregungen und das große Interesse, welches sie meiner Arbeit entgegen gebracht haben.

Für die Schaffung der Rahmenbedingungen im Hause ETAS bin ich dem Geschäftsführer der ETAS GmbH Herrn Dr. Thomas Zurawka, dem General Manager für den Standort Deutschland Herrn Roland Jeutter sowie meinem Abteilungsleiter Herrn Bernhard Ebinger sehr verbunden.

Für die aktive Unterstützung im Hause BOSCH möchte ich Herrn Dr. Klaus-Peter Schnelle sowie Herrn Dr. Weber danken.

Bei meinem Bruder Univ.-Prof. Dr.-Ing. Roland Kasper sowie Herrn Dr. Oliver Schlüter und Herrn Gerhard Stier möchte ich mich für die anregenden Diskussionen bedanken.

Nicht zuletzt danke ich Herrn Arnold Winter, Herrn Ulrich Förch sowie Herrn Eloy Castellanos, die mir stets mit freundlichem Rat und kollegialer Zusammenarbeit zur Seite standen.

Mein Dank gilt darüber hinaus auch meiner Frau Ute und meinen Söhnen Jonas und Lukas, für ihr Verständnis, welches sie mir während der ganzen Zeit entgegen gebracht haben.

KURZFASSUNG

Die ständig steigenden Anforderungen von Gesetzgeber und Kunden an die Kosten, die Sicherheit, die Umweltverträglichkeit sowie den Komfort neuer Produkte, erfordern im Bereich der Fahrzeugtechnik neue Konzepte und Technologien. In diesem Bereich wird daher die Elektrik, Elektronik und Software die Mechanik und Hydraulik immer weiter zurückdrängen oder in einer Symbiose ergänzen und damit zur Schlüsseltechnologie im Fahrzeugbau aufsteigen.

Das Beherrschen dieser neuen Technologien sowie die schnelle Erprobung und anschließende Umsetzung in Produkte, wird zum Erfolgsfaktor für zukunftsfähige Unternehmen. In der vorliegenden Arbeit wird ein zielsystemidentischer Ansatz für das domänenspezifische Rapid Prototyping in der Informations- und Elektrotechnik vorgestellt.

Dabei wird durch die **Partionierung** von Serienstrukturen sowohl der Hardware, als auch der Software, in unabhängige Teilkomponenten sowie der Einführung entsprechender Schnittstellen die **Wiederverwendung** von Erfahrungen und Ergebnissen aus Serienentwicklungen systematisch ermöglicht. Dadurch können, die bislang den Systemlieferanten vorbehaltenen Erfahrungen und Ergebnisse aus Serienentwicklungen in gut handhabbarer Form den Fahrzeugherstellern (OEM) sowie Spezialfirmen für Forschungs- und Entwicklungsaufgaben zur Verfügung gestellt werden, ohne dass die Systemlieferanten ihr Know-How offen legen müssen. Es wird damit eine effiziente Möglichkeit für das **Simultaneous Engineering** zwischen OEM, Systemlieferant und Spezialfirma ermöglicht und somit die OEM in die Lage versetzt, Innovationen selbst im eigenen Unternehmen voranzutreiben und früher als ihre Wettbewerber mit neuen Ideen in den Markt zu gehen.

Des Weiteren kann mit Hilfe der **Wiederverwendung** von Erfahrungen und Ergebnissen aus Serienentwicklungen sehr schnell ein in wesentlichen Punkten mit dem **Zielsystem identischer Prototyp** erstellt und in der realen Umgebung eingesetzt werden. Der Entwicklungsaufwand kann damit auf neue Konzepte und Komponenten begrenzt und somit eine frühe Aussage über Realisierbarkeit und Akzeptanz getroffen werden. Dadurch lassen sich Doppel- und Parallelentwicklungen auf ein Minimum reduzieren und damit die Entwicklungszeit gegenüber traditionellen Verfahren deutlich verkürzen.

Durch den Einsatz **rekonfigurierbarer Logik** wird es ermöglicht, sehr einfach und schnell Änderungen und Erweiterungen in der normalerweise statischen Hardware vorzunehmen. Zusätzlich kann durch Einbetten des Mikrocontrollers in die **rekonfigurierbare Logik** die normalerweise sehr früh zu treffende Entscheidung, ob eine Funktion im Mikrocontroller, in der rekonfigurierbaren Logik oder aufgeteilt auf beide Einheiten realisiert werden soll, noch in sehr fortgeschrittenen Entwicklungsabschnitten beeinflusst werden.

Die in der rekonfigurierbaren Logik umgesetzten Funktionen können z.B. als **ausführbare Spezifikation** für ASIC Entwicklungen verwendet werden. Begünstigt durch die in den letzten Jahren anhaltende Preisreduktion bei rekonfigurierbarer Logik wird es zukünftig evtl. möglich sein, diese direkt im Seriensystem einzusetzen. Dadurch können die heute üblichen ASIC's, welche sowohl digitale, als auch analoge Komponenten beinhalten, als reine analoge ASIC's ausgeführt werden und damit eine für analoge Belange optimale Technologie eingesetzt werden. Die bei der Entwicklung der Anwendersoftware erzielten Ergebnisse können einfach und ohne großen Aufwand im Seriensystem **weiterverwendet** werden, da die Anwendersoftware nur über mit dem Zielsystem identische Schnittstellen auf Hardwaretreiber, Service Routinen sowie das Betriebssystem zugreift.

Durch die Steigerung des Automatisierungsgrades von Entwicklungsprozessen z.B. mit Hilfe der durchgängigen Verwendung von **grafischen Entwicklungswerkzeugen** mit automatischer **Codegenerierung** für Prototypen- und Seriensysteme, können Fehlerquellen reduziert und somit die Qualität verbessert werden. Des Weiteren kann dadurch die Entwicklung beschleunigt werden.

Der in der vorliegenden Arbeit beschriebene zielsystemidentische Ansatz für das domänenspezifische Rapid Prototyping wird am Beispiel eines Entwicklungs-Steuergerätes für Dieselmotoren dargestellt.

ABSTRACT

Requirements for vehicles in terms of cost, driver/passenger safety, low environmental impact and driver/passenger comfort are stringent already and will become more stringent in the future. To fulfill these requirements, new concepts and technologies in vehicle manufacturing technology are needed. Electronics and software solutions will replace mechanical and hydraulic solutions or at least will be combined with the older technologies in the implementation of new functions. In other words, electronics and software will become the key technologies in vehicle manufacturing.

Managing these new technologies in development by means of tests and quality control is a vital factor for successful products and will help determine how successful companies can be now and in the future. This paper introduces a target-system- identical approach to domain-specific rapid prototyping in the fields of information technology and electrical engineering.

The systematic **reuse** of experiences and results gained during the development of production projects is possible by **partitioning** control system hardware and software, as well as by introducing appropriate interfaces. Suppliers will be able to share this yield of hardware and software components with vehicle manufacturers (OEMs) and with companies specializing in specific areas of system development. By sharing these components, suppliers will be able to share their experience and expertise gained through the development of production proven control systems without compromising their proprietary know-how. Thus OEMs, system suppliers and specialized companies can engage in efficient **simultaneous** component **engineering.** This approach will enable OEMs to quickly implement innovations within their own organizations and beat their competitors to the market.

Furthermore, this **reuse** of experiences and results from production development facilitates and accelerates the creation of prototypes that are essentially identical to the target system and can therefore be readily used in the original environment. The development effort can thus be focused on new concepts and components, which in turn will enable a much earlier evaluation of feasibility and acceptance. Thus duplication and parallel development can be reduced to a minimum. Compared to traditional approaches, this approach will reduce overall development time.

By using hardware with **reconfigurable logic,** rather than static hardware, modifications and extensions become possible and are easy and fast. Also, in traditional development, the decision as to where to implement a given function - in the microcontroller, in the reconfigurable logic or distributed between both units - has to be made very early, when not all information is available yet. Embedding the microcontroller in **reconfigurable logic,** however, lets developers postpone this decision until very late in development, when it can be made with much greater confidence.

The functions realized in the reconfigurable logic can be used, for example, as **executable specifications** for ASIC (Application-specific Integrated Circuit) developments. If prices for reconfigurable logic continue to go down, as they have been for years now, it may eventually be possible to use reconfigurable logic directly in production solutions. Then the ASIC's in common use today - comprising digital and analogue components - could be executed as pure ASIC's - comprising analogue components only - and the best technology available for analogue needs could then be used. The results obtained during development of the user software can be **reused** in the production system without much

extra cost, since the user interface software accesses hardware drivers, service routines and the operating system via target system identical interfaces.

Consistently using **graphic development tools** capable of **automatic code generation** for prototype and production systems, as well as other tools that can be used to automate manual, error-prone steps in the development process, will further reduce development time.

This thesis describes target-identical domain-specific rapid prototyping using a development ECU for diesel engines as a sample application.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

Kapitel 1

EINLEITUNG

1 Einleitung

1.1 Anforderungen an Technologien für zukunftsfähige Entwicklungen

Ständig steigende Anforderungen des Gesetzgebers bzw. der Kunden an die Kosten, Sicherheit, Umweltverträglichkeit sowie den Komfort neuer Produkte erfordern neue Konzepte und Technologien, um diese möglichst schnell umsetzen zu können. Davon betroffen sind nahezu alle Industriebereiche wie beispielsweise die Telekommunikations-industrie, Unterhaltungselektronik, Automatisierungs-, Medizin- und Fahrzeugtechnik. Das Beherrschen dieser neuen Technologien sowie die schnelle Umsetzung in Produkte ist die Herausforderung für zukunftsfähige Unternehmen.

Vor allem bei der Fahrzeugtechnik werden Elektrik, Elektronik und Software die Mechanik und Hydraulik immer weiter zurückdrängen oder in einer Symbiose ergänzen und damit zur Schlüsseltechnologie im Fahrzeugbau aufsteigen. Der Wert der Elektrik und Elektronik im Automobil wird von 22% im Jahr 2002 (durchschnittlich 2.250 €) auf 35% (durchschnittlich 3.870 €) im Jahr 2010 steigen. Nahezu jedes Modul im Fahrzeug wird durch diese Technologien »intelligenter« [Mer02].

Der zunehmenden Komplexität der Anwendungen, wie auch der Architekturen, steht ein immer kürzerer Produktzyklus und damit eine sinkende Entwicklungszeit gegenüber. Diese Herausforderung ist nur durch eine extreme Steigerung der Produktivität in der Entwicklung zu bewältigen. Eine solche Produktivitätssteigerung kann z.B. durch eine drastische Steigerung des Automatisierungsgrades, eine wesentliche Verbesserung der verwendeten Methoden, durch systematische Wiederverwendung großer Systemteile und durch Vermeidung von Mehrfachentwicklungen erreicht werden.

Die Wiederverwendung von Systemteilen kann sowohl auf Seiten der Software, als auch auf Seiten der Hardware erfolgen. Im Bereich der Software können das beispielsweise Entwicklungswerkzeuge, Betriebssysteme, low-level Treiber, Kommunikationsprotokolle oder auch größere Teile der Anwendersoftware sein. Bei der Hardware kann man auf bekannte Bauelemente, komplette Schaltungsteile auf Schaltplan- oder Layoutebene oder sogar auf komplette Leiterplatten zurückgreifen. Durch den immer breiteren Einsatz von rekonfigurierbaren Logik-Bausteinen kann auch eine Wiederverwendung auf der Ebene von Hardwarebeschreibungssprachen (HDL) oder Intellectual Property (IP) Cores statt-finden.

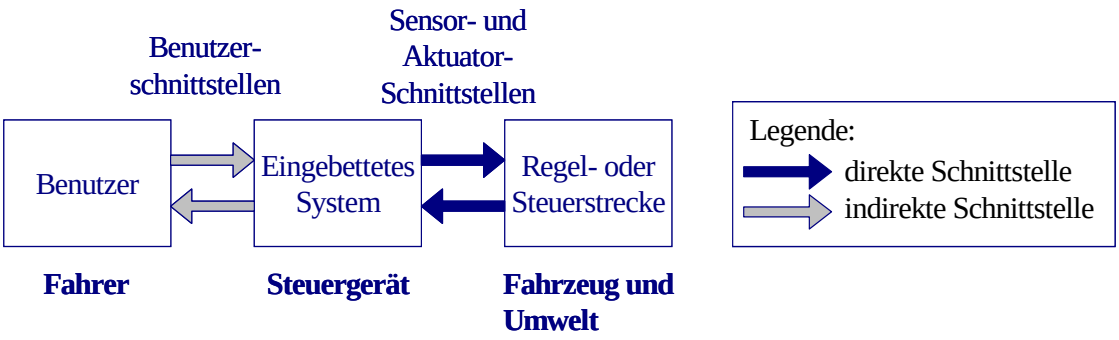
Um Doppel- und Parallelentwicklungen zu reduzieren ist es beispielsweise erforderlich, dass die durch Simulation nachgewiesene neue Funktionalität so früh wie möglich in der realen Umgebung auf Akzeptanz und tatsächliche Realisierbarkeit überprüft werden kann. So ist es heute üblich, mehrere Ansätze soweit parallel zu verfolgen, bis sie am realen Prozess evaluiert werden können. Man spricht dann auch häufig von einer „erlebbaren Spezifikation“. Je früher es demzufolge möglich ist, eine Applikation am realen Prozess zu erproben, desto früher kann die Entscheidung über die weiter zu verfolgende Strategie gefällt und somit ein nicht unerhebliches Maß an Doppelentwicklung vermieden werden.

Um diese Aufgaben kostengünstig und in der zur Verfügung stehenden Zeit lösen zu können, muss man immer häufiger neue **eingebettete, mikroelektronische oder mechatronische Ansätze** erarbeiten.

Im Weiteren wird kurz auf die Definition und die Besonderheiten eingebetteter und mechatronischer Systeme eingegangen.

1.2 Definition eingebetteter Systeme

Nach [Ern99] versteht man unter einem eingebetteten System ein Computersystem, das in ein technisches System „eingebettet“ ist, das selbst aber gegenüber dem Anwender nicht als Computer in Erscheinung tritt. Beispielsweise hat der Fahrer eines Fahrzeuges häufig nur mittelbaren, oft nur geringen und teilweise auch gar keinen Einfluss auf die Funktion der Steuergeräte. Die Benutzerschnittstellen sind fast immer indirekt durch Schalter und Pedale realisiert (Abbildung 1-1).



Quelle: [SZ03]

Abbildung 1-1: Merkmale eingebetteter Systeme

Der Entwurf sowie die Hardware- und Softwarearchitekturen eingebetteter Systeme unterscheiden sich wesentlich von denen der klassischen Datentechnik. Eingebettete Systeme sind auf den jeweiligen Anwendungsfall in möglichst vielen Punkten (Ein- / Ausgänge, Rechenleistung, Bauform, Umweltbedingungen, etc.) optimal angepasst. Während es anfangs noch üblich war, ältere und damit kostengünstigere Prozessoren aus der Datentechnik weitgehend unverändert zu übernehmen, lässt sich heute bei eingebetteten Systemen eine Diversifizierung in eine Vielzahl von spezialisierten Architekturen beobachten. Dabei wird ein sehr breites Spektrum von 4, 8, 16 und 32 Bit Mikrocontrollern und digitalen Signalprozessoren (DSP) abgedeckt. Komplexere eingebettete Systeme bilden meist heterogene Architekturen aus Mikrocontrollern, DSP, Coprozessoren, analogen und digitalen kundenspezifischen Schaltungen und Speichern (SRAM, FLASH) [Ern99].

1.3 Definition mechatronischer Systeme

Das Kunstwort „Mechatronik“ setzt sich aus „Mechanik“ und „Elektronik“ zusammen und wurde schon 1969 von dem Japaner Ko Kikuchi geprägt. Er hat es zunächst nur im Zusammenhang mit einer damals neuen Generation von Robotern verwandt, die erstmals elektronische Baugruppen integrierten. Im Zeitraum von 1971 bis 1982 war „Mechatronik“ als Handelsname geschützt [VDI03], [Koc00].

Bei mechatronischen Systemen wird die Synergie aus dem Zusammenwirken der klassischen Ingenieurwissenschaften Maschinenbau, Elektrotechnik und Informationstechnik genutzt (Abbildung 1-2).

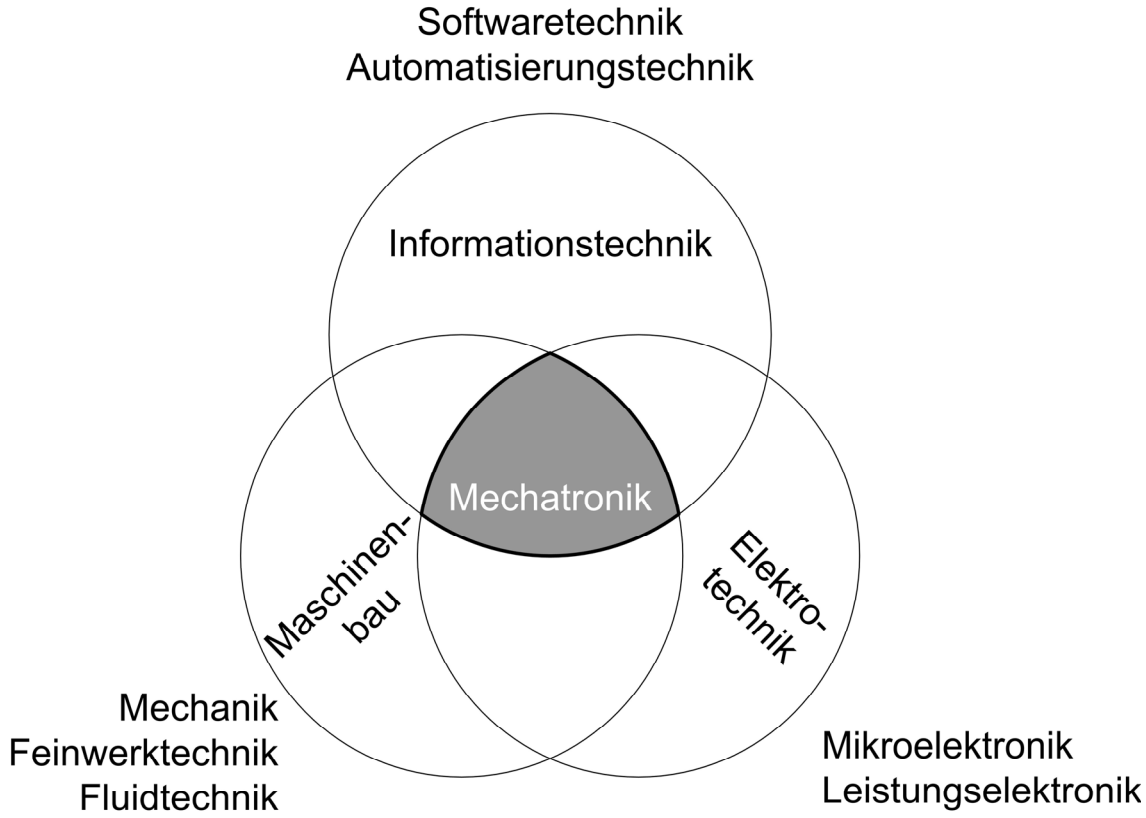


Abbildung 1-2: Fachdisziplinen mechatronischer Systeme

1.3.1 Aufbau mechatronischer Systeme

Der grundsätzliche Aufbau mechatronischer Systeme ist durch ein meist mechanisches Grundsystem, Sensoren, einer Informationsverarbeitung und Aktoren geprägt. Die Komponenten sind grundsätzlich durch Stoff-, Energie- und Informationsflüsse miteinander verkoppelt. Ziel ist es in der Regel, den Stoff- und Energiefluss in dem Grundsystem zu kontrollieren. Dazu werden über Sensoren Informationen über Zustandsgrößen in dem Grundsystem und in der Umgebung detektiert. Die Informationen werden in für die Weiterverarbeitung geeignete Größen umgesetzt und ggf. vorverarbeitet und einer Informationsverarbeitung zugeführt, die mit Funktionen der Digitalelektronik und Software die Sensorinformationen auswertet. Über Steuerungs- und Regelungsalgorithmen werden die notwendigen Informationen für die Einwirkung auf das Grundsystem bestimmt. Nachfolgend werden diese Informationen in entsprechende Größen umgesetzt und ggf. verstärkt, die dann über Aktoren auf das Grundsystem einwirken. Die Einwirkungen sind meist mechanischer Natur, was auch elektromechanische, pneumatische oder hydraulische Wirkprinzipien umfasst. Der grundsätzliche Aufbau mechatronischer Systeme ist in Abbildung 1-3 dargestellt.

Quelle: [GL00]

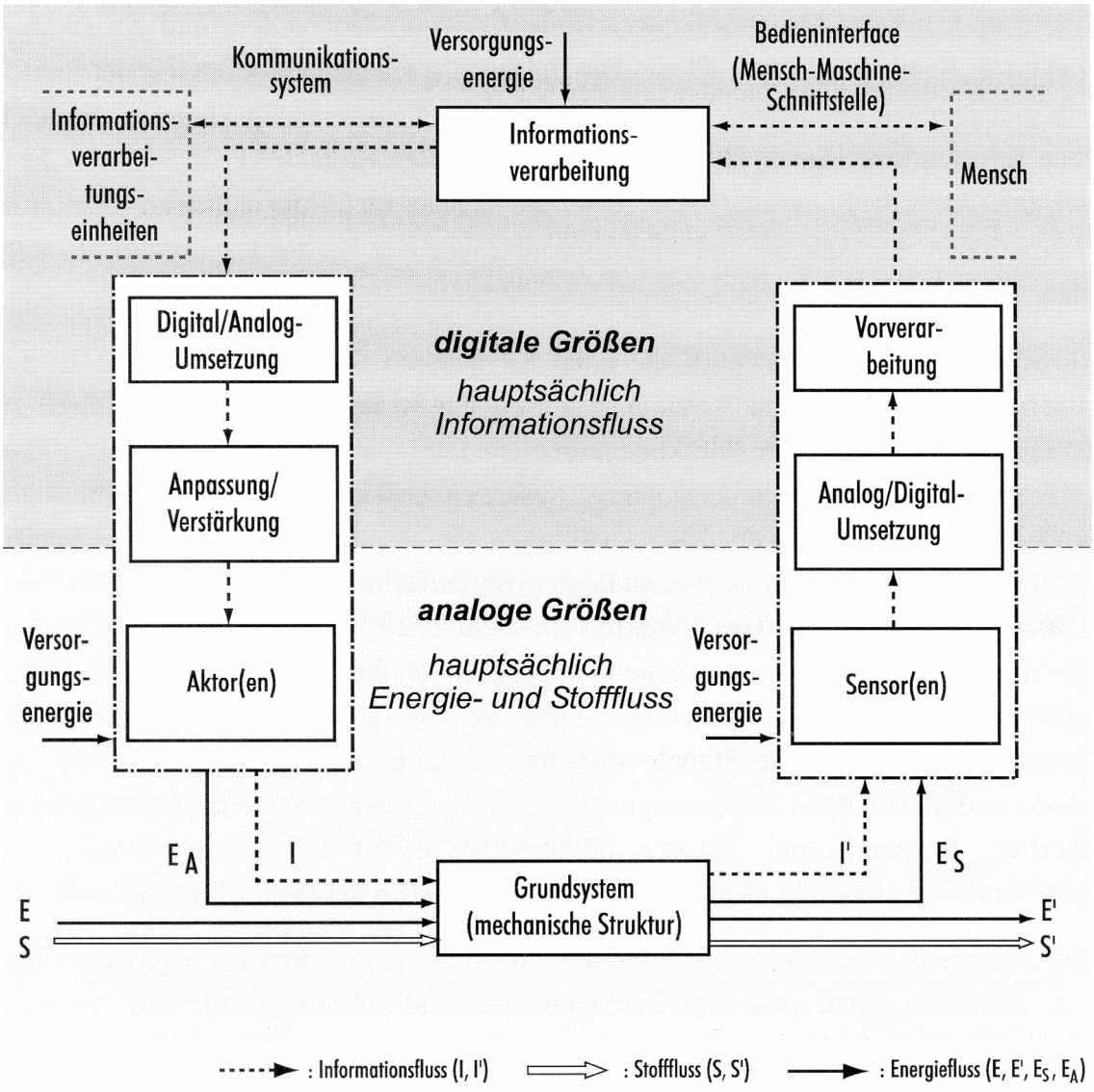


Abbildung 1-3: Grundsätzlicher Aufbau mechatronischer Systeme

Aktoren und Sensoren werden meist durch eine von außen zugeführte Energie versorgt. Bei den Aktoren wird dabei die Intensität des Energiezuflusses durch das Signal der Informationsverarbeitung gesteuert. Die Informationsverarbeitung kann in Verbindung zu weiteren parallelen sowie über- oder untergeordneten Informationsverarbeitungen stehen. Zudem kann die Informationsverarbeitung eine Mensch-Maschine-Schnittstelle aufweisen [Fla02], [GL00].

1.3.2 Beispiele mechatronischer Systeme

Die Mechatronik wird bereits in Entwicklungen und Produkten unterschiedlicher Industriebereiche sehr erfolgreich eingesetzt. Nachfolgend sind beispielhaft einige Anwendungen aufgeführt.

Antriebstechnik:

- Kreiselpumpe [KM00]
- Mehrkoordinatenantrieb [Sch96]
- Autofokus bei Kameras

Computertechnik:

- CD / DVD Laufwerke [EDG01]
- Laser- und Tintenstrahldrucker
- Scanner

Fahrzeugtechnik:

- Aktives Fahrwerk und Fahrzeugfederung [Frü00], [Kau02], [SR02]
- Fahrzeugbremssysteme [Ber02], [Sch00]
- Steer-by-wire [KWG+00], [OBH+02], [Zim02]
- Motormanagement [MI02], [ZL02]
- Getrieberegelung [PH02], [MH02]
- Piezoelektrisches Hydraulikventil (VDI Berichte 1533)

Medizintechnik:

- Positronen-Emissions-Tomographie [SPZ+01]
- Knochenbruchheilung [BRK01]

Die Entwicklung mechatronischer Produkte ist derzeit auf Anwendungen mit großen Stückzahlen sowie Firmen mit entsprechend großem Budget beschränkt. Die Ursachen für diese Beschränkungen liegen unter anderem an den derzeit sehr komplexen Entwicklungsprozessen und den damit verbundenen hohen Entwicklungskosten.

Die Entwicklung eines abgestimmten Zusammenspiels mechanischer, elektronischer und informationstechnischer Komponenten verlangt eine ganzheitliche, disziplinübergreifende Vorstellung zur Konzeption und Ausarbeitung eines optimal ausgelegten Produktverhaltens. Dies erfordert meist einen großen Aufwand hinsichtlich der Modellbildung des übergreifenden Produktverhaltens. Fehlende Forschungs- und Entwicklungsressourcen verhindern aber gerade bei kleinen und mittelständischen Unternehmen den notwendigen Aufbau von Know-How an Methoden und im Umgang mit Entwurfswerkzeugen [Fla02].

Kapitel 2

ZIELSTELLUNG DER ARBEIT

2 Zielstellung der Arbeit

Die Anzahl und Verbreitung motorisierter Fahrzeuge hat in den vergangenen Jahren immer schneller zugenommen. Fahrzeuge sind aus unserem täglichen Leben nicht mehr wegzudenken. Aber gerade mit dieser immer weiteren Verbreitung vergrößert sich auch die Anzahl der Randbedingungen und Anforderungen an neue Fahrzeuge. In Abbildung 2-1 sind einige wichtige Gruppen dargestellt, welche Einfluss auf die Entwicklung moderner Fahrzeuge nehmen.

Quelle: [Kas04]

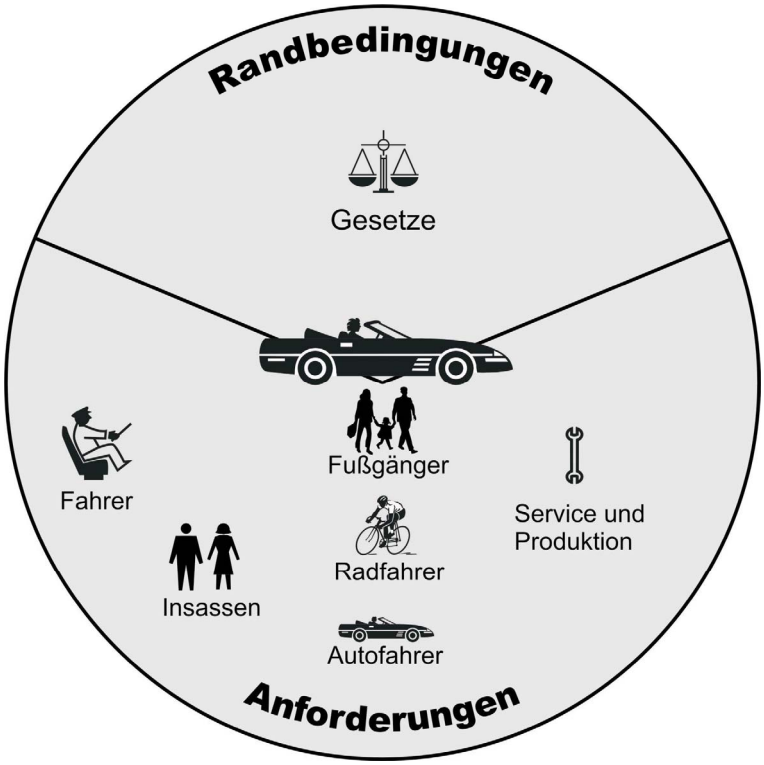


Abbildung 2-1: Randbedingungen und Anforderungen für die Fahrzeugindustrie

Die Anforderungen der einzelnen Gruppen können dabei sehr unterschiedlich sein. Folgende Anforderungen und Randbedingungen haben sich als besonders wichtig herausgestellt:

- Steigerung der Umweltverträglichkeit
  - Abgasbestimmungen (USA: LEV II, Europa: EURO IV)
  - On-Board Diagnose (USA: OBD II, Europa: EOBD)
  - Verbrauch (Steuern, Umweltbewusstsein)
- Steigerung der Sicherheit
  - ABS, ESP, ACC, AFS, etc.
- Steigerung des Komforts
  - Klimaanlage, Navigationssystem, Spiegel- und Sitzverstellung, etc.
- Steigerung der Verfügbarkeit (Reduzierung der 100.000 km Ausfälle, etc.)
- Reduzierung der Kosten für Entwicklung, Produktion und Betrieb

Um die Vielzahl der ständig steigenden Anforderungen und schärfer werdenden Randbedingungen schnell und effizient in neue Produkte umsetzen zu können, sind an

diese Situation angepasste Entwicklungsmethoden erforderlich. Forderungen an diese Methoden sind unter anderem:

- Verkürzung der Entwicklungszeiten,
- Wiederverwendbarkeit von Ergebnissen,
- frühe Aussage über Realisierbarkeit,
- weit reichende Änderungs- und Erweiterungsmöglichkeiten,
- einfache Darstellbarkeit kundenspezifischer Lösungen unter Berücksichtigung von Varianten und Know-How Schutz,
- gute Debugmöglichkeiten.

Wie unter anderem aus der Mercer Studie „Automobiltechnologie 2010“ [Mer02] hervorgeht, werden sich die meisten Innovationen in naher Zukunft vorwiegend durch Fortschritte im Bereich Elektrotechnik und Informationstechnik realisieren lassen. Es wird dementsprechend eine heute bereits einsetzende Verschiebung von traditionellen mechanischen Lösungsansätzen zu durch die Elektrotechnik und Informationstechnik geprägten Ansätzen geben.

Speziell im Fahrzeugbau, wo es traditionell eine enge Zusammenarbeit zwischen den Fahrzeugherstellern und den Zulieferern gibt, führt diese Verschiebung von zukünftig teilweise wettbewerbsentscheidenden Entwicklungskompetenzen zu einem Umdenken. In der Vergangenheit konzentrierten sich die Fahrzeughersteller vorwiegend auf die oberste Systemebene und somit auf die Spezifikation und Integration von Subsystemen. Sie waren zwar an der Spezifikation der Subsysteme beteiligt, die Kompetenz, diese zu entwickeln, lag aber ausschließlich bei den Zulieferern. Diese Verteilung der Kompetenzen kann man anschaulich anhand eines V-Modells darstellen (Abbildung 2-2). Weitere Informationen zum V-Modell sind in Abschnitt 3.1.2 zu finden.

Quelle: [Kas04]

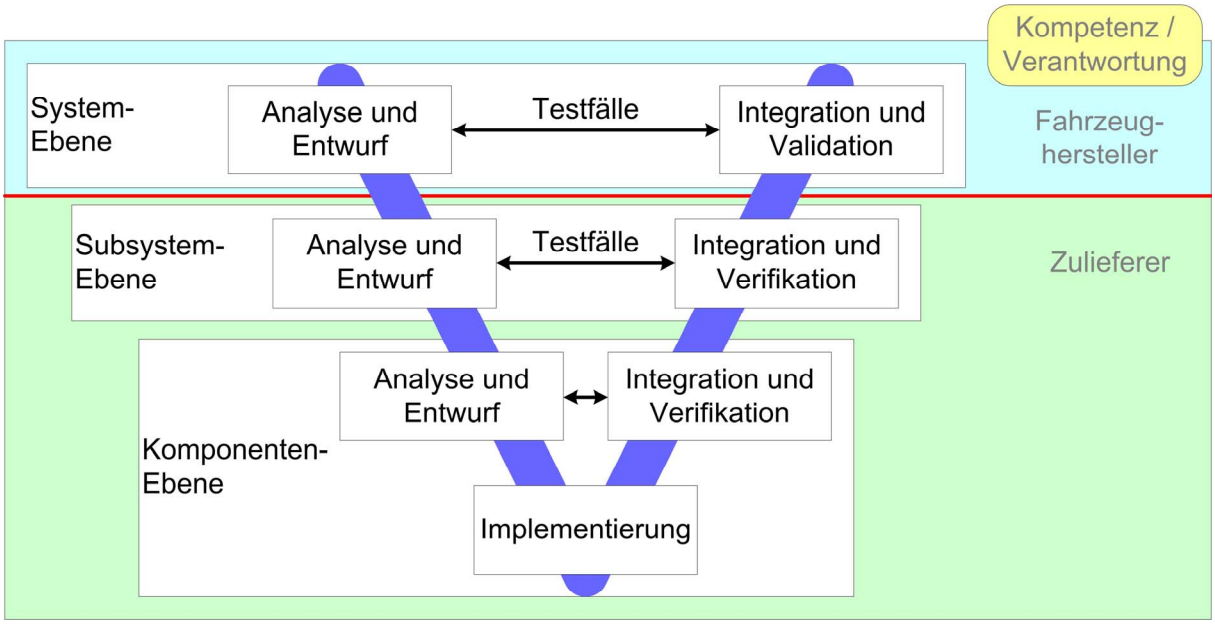


Abbildung 2-2: bisherige Kompetenzverteilung

Um zukünftige Innovationen selbst im eigenen Unternehmen vorantreiben zu können und somit früher als die Wettbewerber mit neuen Ideen auf den Markt zu kommen, wird von den Fahrzeugherstellern eine Aufteilung der Kompetenzen nach Abbildung 2-3 angestrebt. Darin übernehmen die Zulieferer vorwiegend die Rolle eines Komponentenzulieferers bzw.

des Zulieferers eines offenen Subsystems, welches vom Fahrzeughersteller bei Bedarf noch an seine individuellen Bedürfnisse angepasst werden kann.

Quelle: [Kas04]

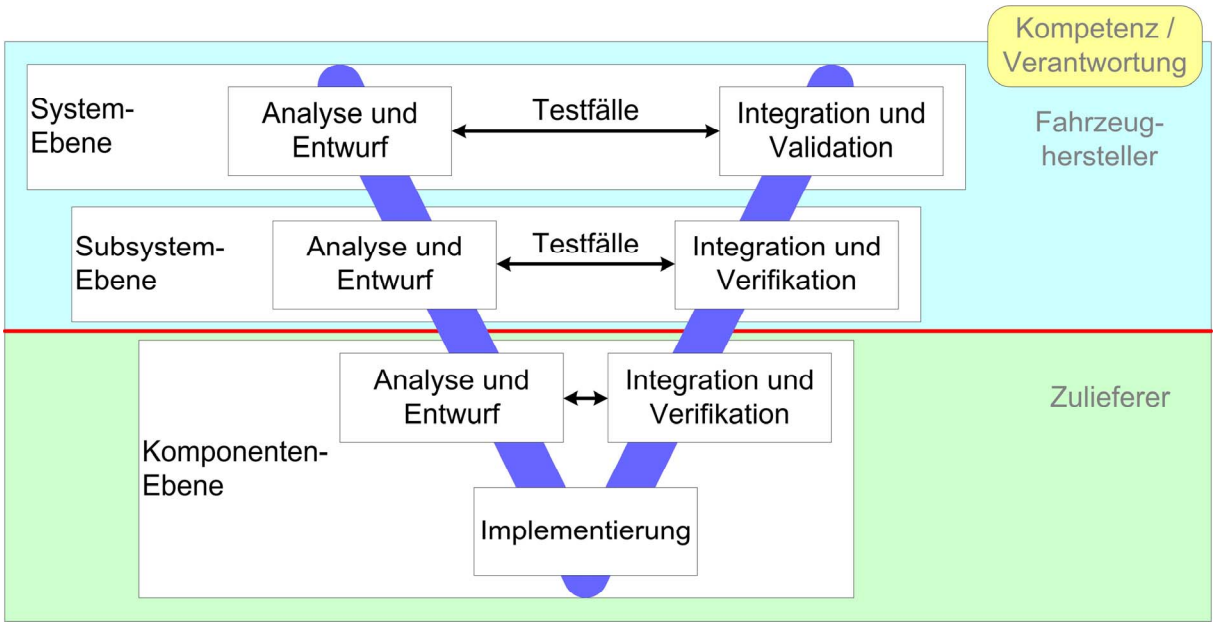


Abbildung 2-3: zukünftige Kompetenzverteilung

Eine noch weitergehende Forderung bei der Einbindung von Entwicklungspartnern wurde in [VDI03] formuliert. Da - wie die Erfahrungen der letzten Jahre gezeigt haben - gerade etablierte Systemlieferanten wenig Bestreben zeigen, bewährte Konzepte durch neue Lösungen zu ersetzen, wird ein Vorgehensmodell nach Abbildung 2-4 vorgeschlagen.

[VDI03]

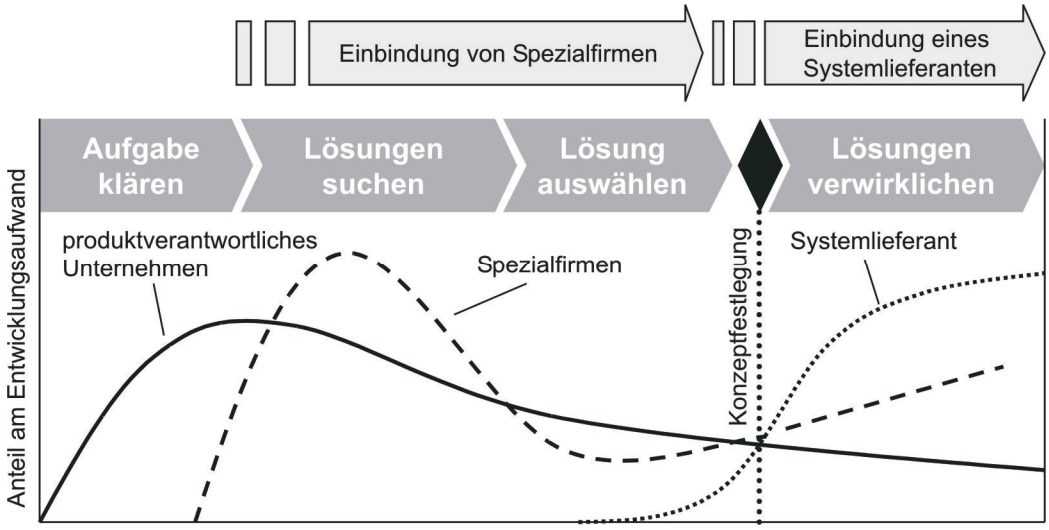


Abbildung 2-4: Einbindung von Entwicklungspartnern

Bei diesem Vorgehensmodell wird bewusst auf die Einbindung eines Systemlieferanten bis zur Fertigstellung eines Konzepts für das betrachtete System verzichtet. Durch die direkte Zusammenarbeit mit Firmen, welche spezielles Know-How bezüglich des Einsatzes bestimmter Technologien haben, wird nach der für den Hersteller besten und günstigsten Lösung gesucht. Nachdem die Entscheidung für ein bestimmtes Konzept gefallen ist, wird ein Systemlieferant ausgewählt und die Verantwortung für die weitere Entwicklung an diesen übertragen.

Die vorliegende Arbeit konzentriert sich auf die Optimierung der domänenspezifischen Entwicklung um damit schnell und einfach Labormuster, Funktionsmuster und Vorserienprodukte realisieren zu können. Dazu wird das in [VDI03] (siehe Abbildung 3.2) vorgeschlagene V-Modell welches die Domänen Maschinenbau, Elektrotechnik und Informationstechnik berücksichtigt auf die Domänen Elektrotechnik und Informationstechnik eingeschränkt, da dort zukünftig die meisten Innovationen erwartet werden. Des Weiteren sind Änderungen an mechanischen Komponenten meist sehr aufwändig was zu hohen Entwicklungskosten und langen Enzwicklungszeiten führen würde. Zudem verfügen die Fahrzeughersteller meist nicht über die entsprechenden Einrichtungen und Kompetenzen um solche Änderungen vorzunehmen. Die Optimierung der domänenspezifischen Entwicklung soll vor allem durch die Bereitstellung eines Rapid-Prototyping-Entwicklungs-systems, bestehend aus Software und Hardware, erreicht werden. Das neu definierte Entwicklungssystem soll unter Berücksichtigung der eingeführten Standards eine wesentlich größere Flexibilität bietet. Dieses Entwicklungssystem soll des Weiteren die normalerweise nur dem Systemlieferanten vorbehaltenen „inneren“ Eigenschaften und Möglichkeiten dem Fahrzeughersteller und den Spezialfirmen zugänglich machen. Modifikationen sollen in der Software, der Mikrocontroller Hardware, der digitalen Elektronik sowie der analogen Elektronik möglich sein. Das soll realisiert werden, ohne dass der Systemlieferant sein komplettes Know-how offen legen muss. Der Fahrzeughersteller bzw. die Spezialfirmen müssen aber dennoch in die Lage versetzt werden, Modifikationen ohne die Unterstützung des Systemlieferanten vorzunehmen, da dieser die Kapazitäten für eine solche Unterstützung normalerweise nicht oder nur mit einer gewissen Verzögerung bereitstellen kann.

Der Vorteil eines solchen Vorgehens für den Fahrzeughersteller bzw. die Spezialfirmen liegt darin, dass sie relativ schnell in der Lage sind, ein erstes Labormuster aufzubauen und dann konkret an ihrer eigentlichen Aufgabe arbeiten können. Bei den Teilen, an denen (vorerst) nichts geändert werden soll oder muss, werden die vom Systemhersteller zur Verfügung gestellten Teile verwendet. Für den Systemhersteller besteht der Vorteil darin, dass er das vom Fahrzeughersteller bzw. den Spezialfirmen überarbeitete Subsystem relativ einfach wieder übernehmen, weiterentwickeln und in eine serientaugliche Lösung überführen kann.

Ziel dieser Arbeit ist es, Lösungswege für diese Anforderungen herauszuarbeiten und diese exemplarisch an einem Entwicklungssteuergerät für Dieselmotoren unter Beweis zu stellen. Im Folgenden sind die wichtigsten Forderungen an einen Ansatz zur Optimierung der domänenspezifischen Entwicklung im Bereich Elektrotechnik und Informationstechnik zusammengestellt:

- **Kurze Entwicklungszeiten**
  Durch die Bereitstellung einer breiten Basisfunktionalität soll es sehr schnell möglich sein, ein erstes Prototypen-System aufzubauen.
- **Sehr weit reichende Änderungs- und Erweiterungsmöglichkeiten**
  Änderungen und Erweiterungen sollen an möglichst vielen Stellen sowohl der Hardware, als auch der Software problemlos und schnell durchgeführt werden können. Das Debugging der Hard- und Software sollte dabei ebenfalls berücksichtigt werden. Über offene Schnittstellen sollte es möglich sein, Ergebnisse anderer Arbeiten einzubinden.
- **Flexibel einsetzbar**
  Das System sollte so flexibel ausgelegt werden, dass es nicht nur speziell für ein Anwendungsgebiet eingesetzt werden kann, sondern im Bedarfsfall durch geringe Modifikationen in vielen Anwendungsgebieten verwendet werden kann.

- **Ausreichend Ressourcen (Rechenleistung, Speicher, I/O, etc.)**
  Der Anwender sollte sich bei der Entwicklung seiner Funktionalität keine Gedanken über Rechenleistung, Speicher und I/O-Ressourcen machen müssen. Eine Optimierung auf diese Aspekte sollte erst in einem zweiten Schritt erfolgen müssen.
- **Verhalten identisch mit dem des Zielsystems**
  Das System sollte sich an den Schnittstellen bezüglich des elektrischen und zeitlichen Verhaltens so weit als möglich identisch zu dem angestrebten Zielsystem verhalten. Allerdings nur soweit es die Flexibilität nicht zu sehr einschränkt.
- **Umgebungsbedingungen vergleichbar mit denen des Zielsystems**
  Das System sollte für Umgebungsbedingungen (Temperatur, EMV, etc.) vergleichbar denen des Zielsystems ausgelegt sein.
- **Einfache Weiterverwendbarkeit der Ergebnisse**
  Die erzielten Entwicklungsergebnisse sollten einfach und ohne große Risiken im späteren Zielsystem weiterverwendet werden können.

Kapitel 3

STAND DER WISSENSCHAFT UND TECHNIK

3 Stand der Wissenschaft und Technik

In diesem Abschnitt werden eingeführte Vorgehensmodelle bei der Entwicklung mechatronischer Systeme sowie Entwurfs- und Konstruktionsmethoden dargestellt. Anschließend wird auf speziell für die Fahrzeugelektronik wichtige Standards für die Entwicklung eingegangen.

3.1 Vorgehensmodelle zur Entwicklung mechatronischer Systeme

Zur Entwicklung eingebetteter und mechatronischer Systeme werden unterschiedliche Vorgehensmodelle verwendet. Als wichtige Vertreter seien hier das Wasserfall-, das Spiral- und das V-Modell genannt. Im Folgenden wird auf diese Vorgehensmodelle kurz eingegangen.

3.1.1 Wasserfall-Modell

Das ursprünglich aus der Softwareentwicklung bekannte Wasserfall-Modell schlägt vor, die Entwicklung in aufeinander folgenden Phasen durchzuführen. In Abbildung 3-1 wird die Funktionsentwicklung nach dem Wasserfall-Modell dargestellt. Die einzelnen Entwicklungsschritte sind dabei [SZ03]:

1. **Analyse**
   Festlegung des Funktionsnetzwerks, der Schnittstellen, der Funktionen und der Kommunikation zwischen den Funktionen des Systems oder Teilsystems.
2. **Spezifikation**
   Beschreibung der „idealen“ Funktionen. Das bedeutet ohne Berücksichtigung der Randbedingungen der späteren Realisierung wie beispielsweise die Begrenzung von Ressourcen.
3. **Design**
   Beim Design werden die „realen“ Randbedingungen der Realisierung wie beispielsweise die Begrenzung von Ressourcen berücksichtigt. Die Software-Architektur einschließlich der zu verwendenden Standards werden ebenfalls in diesem Schritt festgelegt.
4. **Implementierung**
   Realisierung der zuvor definierten Funktionen (z.B. Erstellen von Software).
5. **Integration**
   Zusammenführen der realisierten Teilfunktionen, um die Gesamtfunktionalität zu realisieren.
6. **Inbetriebnahme** und **Test**
   Überprüfen der Gesamtfunktionalität.
7. **Kalibrierung**
   Feinabstimmung auf das reale System.
8. **Produktion**
   Vervielfachen der Ergebnisse und Sicherstellen der zuvor abgestimmten Eigenschaften.
9. **Service**
   Diagnose und Fehlersuche im Laufe der Lebensdauer des Systems.

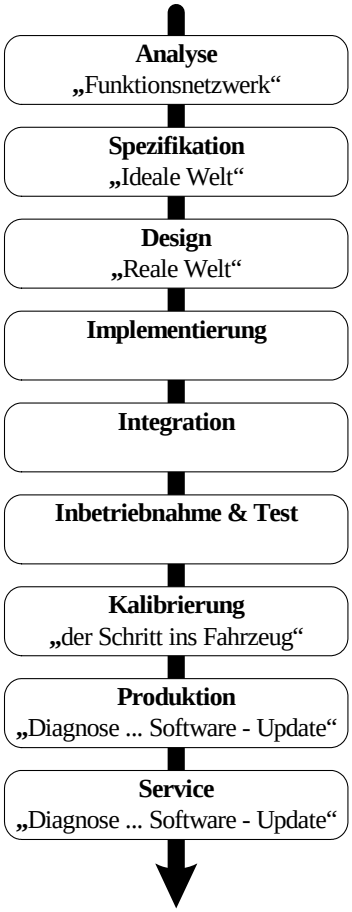


Abbildung 3-1: Funktionsentwicklung nach dem Wasserfall-Modell

3.1.2 V-Modell

Das V-Modell ist eine Weiterentwicklung des Wasserfall-Modells. Diese Weiterentwicklung besteht darin, dass die Qualitätssicherung mit integriert wurde. Die Integration der Qualitätssicherung wird dadurch erreicht, dass bereits beim Systementwurf Anwendungs- und Testfälle festgelegt werden. Die Eigenschaften des Systems werden anhand dieser Anwendungs- und Testfälle schon während der Entwicklung laufend abgesichert. Dieses Vorgehensmodell lässt sich am besten in Form eines „V“ darstellen, (Abbildung 3-2) was diesem Modell auch den Namen verliehen hat. [GM03], [SZ03], [VDI03], [IABG]

Quelle: [VDI03]

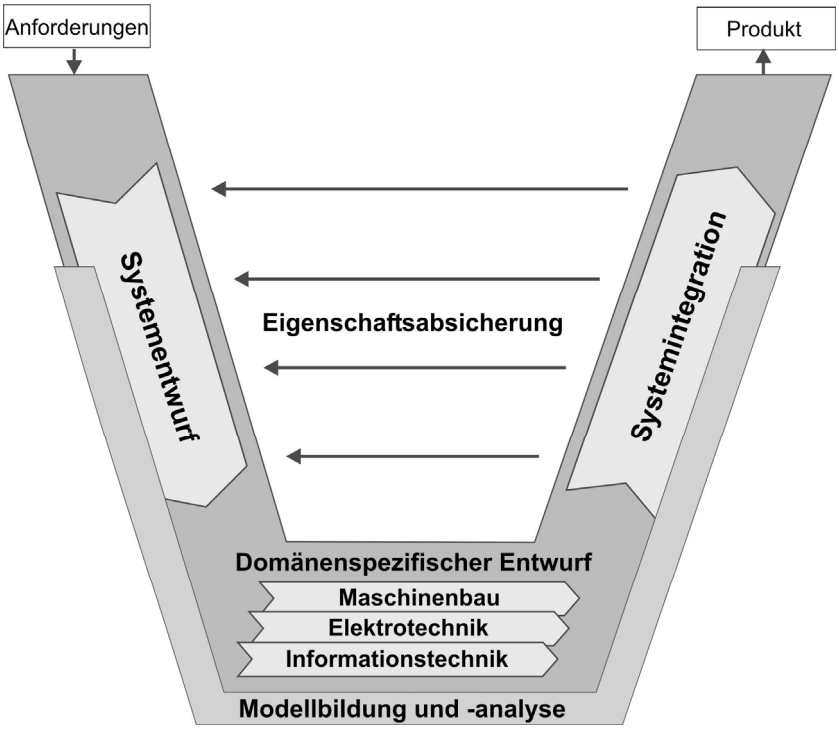


Abbildung 3-2: Das V-Modell

1. **Anforderungen**
   Aufgabenstellung, welche das zu entwickelnde System erfüllen soll. Die Bewertung des Systems findet ebenfalls anhand dieser Anforderungen statt.
2. **Systementwurf**
   Festlegung eines domänenübergreifenden Lösungskonzepts. Hierzu wird die Gesamtfunktion des Systems unter Berücksichtigung der späteren Zusammenführung in relevante Teilfunktionen zerlegt.
3. **Domänenspezifischer Entwurf**
   Basierend auf dem entworfenen Lösungskonzept erfolgt die Auslegung und Realisierung der Funktionen meist getrennt für die beteiligten Domänen.
4. **Systemintegration**
   Die Ergebnisse der einzelnen Domänen werden zu einem Gesamtsystem zusammengeführt.
5. **Eigenschaftsabsicherung**
   Die Eigenschaften des Systems werden anhand der Anwendungs- und Testfälle laufend abgesichert.
6. **Produkt**
   Ergebnis eines komplett durchlaufenen V-Modells.

Ein komplexes Produkt entsteht in der Regel nicht in einem einzigen Durchlauf. Es sind mehrere Durchläufe erforderlich, wobei der Reifegrad des Produkts mit jedem Durchlauf steigt (Abbildung 3-3).

Quelle: [VDI03]

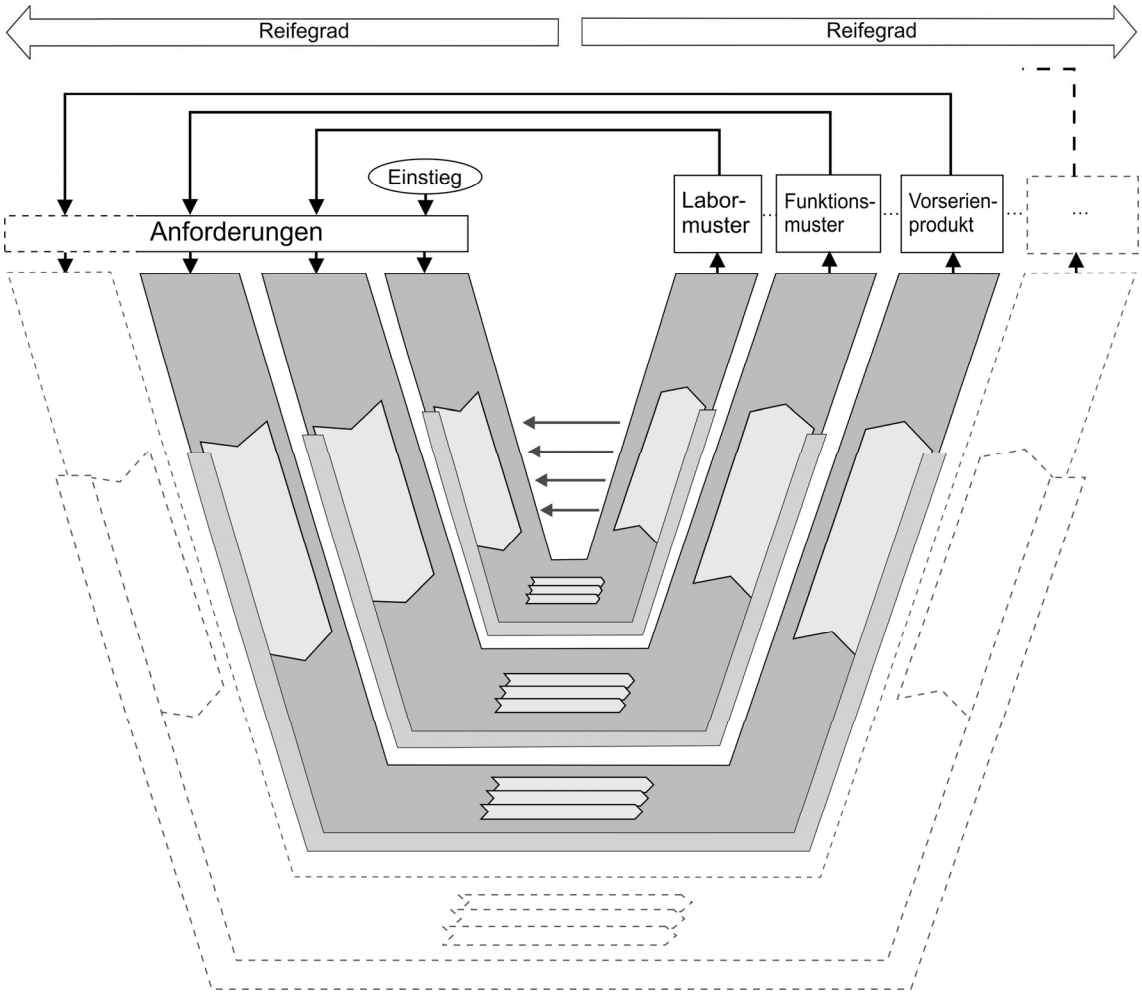


Abbildung 3-3: Mehrfach durchlaufenes V-Modell

In einem ersten Zyklus wird beispielsweise das Produkt funktional spezifiziert, erste Wirkprinzipien und Lösungselemente ausgewählt und grob dimensioniert, im Systemkontext auf Konsistenz geprüft und exemplarisch realisiert. Das Ergebnis ist in der Regel ein Labormuster. Dieses wird in weiteren Zyklen konkretisiert, um erste Funktionsmuster zu erstellen. Je nach Entwurfsfortschritt, Art und Komplexität der Entwicklungsaufgabe sind ggf. weitere Zyklen erforderlich, um zum serienreifen Produkt zu gelangen.

3.1.3 Spiral-Modell

Das ebenfalls aus der Softwareentwicklung bekannte Spiral-Modell (Abbildung 3-4) verfolgt wie das mehrfach durchlaufene V-Modell einen inkrementellen Ansatz. Es wird zunächst ein Prototyp mit eingeschränkter Funktionalität entwickelt, der in der realen Umgebung eingesetzt und erprobt werden kann. Damit ist es möglich, Defizite schon in einer frühen Phase zu erkennen. Diese Informationen werden verwendet, um einen verbesserten Prototypen zu erstellen, der in einer weiteren Iteration validiert wird. Dieser evolutionäre Prototypenzyklus wird wiederholt bis alle Anforderungen erfüllt sind [SZ03].

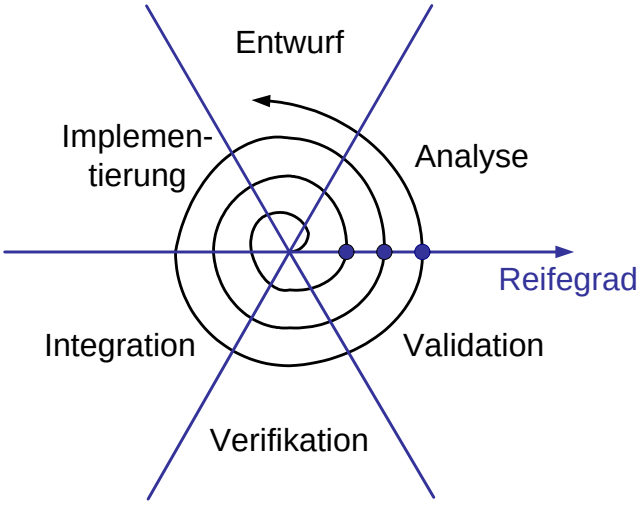


Abbildung 3-4: Spiral-Modell

Die von Iteration zu Iteration wiederkehrenden Aufgaben sind jeweils unter dem gleichen Winkel im Koordinatensystem zu finden. Der Schnittpunkt eines jeden Umlaufs mit der rechts vom Startpunkt liegenden Abszisse, entspricht einem Prototyp des zu entwickelnden Systems. Mit jedem neuen Prototyp steigt der Reifegrad des Systems.

3.1.4 Entwicklungsmodell für mechatronische Systeme nach [VDI03]

In der Richtlinie [VDI03] wird ein flexibles Vorgehensmodell basierend auf folgenden drei Elementen vorgeschlagen.

1. **Problemlösungszyklus als Mikrozyklus**
   Die Strukturierung des Vorgehens im Entwicklungsprozess erfolgt dabei auf Grundlage eines allgemeinen Problemlösungszyklus. Durch Aneinanderreihen und Verschachteln von Vorgehenszyklen lässt sich die Prozessplanung flexibel an die Eigenheiten jeder Entwicklungsaufgabe anpassen.

2. **V-Modell als Makrozyklus**
   Das V-Modell ist eine Art Richtschnur welche die Abfolge wesentlicher Teilschritte bei der Entwicklung beschreibt. Weitere Informationen sind unter anderem in Abschnitt 3.1.2 zu finden.

3. **Prozessbausteine für wiederkehrende Arbeitsschritte**
   Für einige bei der Entwicklung wiederkehrende Aufgabenstellungen kann die Bearbeitung in Form von teilweise vordefinierten Prozessbausteinen beschrieben werden.

3.2 Modellbasierter Systementwurf

Bei der modellbasierten Entwicklung wird mit Hilfe von Modellen der beteiligten Komponenten ein domänenübergreifender, rechnerunterstützter Entwurf mechatronischer Systeme ermöglicht. Die bei den unterschiedlichen Fachdisziplinen entstandenen Darstellungsformen für Modelle sind in der Regel nicht kompatibel. Da eine Beurteilung des Systems aber erst nach Betrachtung des Zusammenwirkens aller beteiligten Fachdisziplinen sinnvoll möglich ist, werden häufig mathematische Beschreibungen für die Modelle verwendet. Die Mathematik wird aufgrund ihrer Allgemeingültigkeit von allen Fachdisziplinen anerkannt. Während der Systementwicklung werden in der Regel mehrere auf die jeweilige Aufgabe zugeschnittene Modelle der einzelnen Komponenten erstellt. Diese Modelle entstehen normalerweise durch Verfeinerung der Modelle früherer Entwicklungsphasen und unterscheiden sich beispielsweise durch ihren Detaillierungsgrad.

Das Vorgehen beim modellbasierten Entwickeln ist beispielhaft in Abbildung 3-5 dargestellt. Weitere Details sind in [VDI03] zu finden, der auch das folgende Bild sowie die Erläuterungen entnommen sind.

1. **Zielformulierung**
   Bei der Zielformulierung werden zunächst die Untersuchungsziele sowie Aufgaben festgelegt, damit die geeigneten Methoden zur Modellbildung ausgewählt werden können. Solche Ziele bzw. Aufgaben könnten beispielsweise Prinzipuntersuchungen, Reglerauslegung, Optimierung bestehender Systeme, etc. sein. [Ber01]

2. **Modellbildung**
   Um das Verhalten mit der erforderlichen Genauigkeit beschreiben zu können, werden Modelle auf verschiedenen Abstraktionsebenen gebildet. Gebräuchliche Modellierungsebenen sind dabei topologische, physikalische, mathematische und numerische (Abbildung 3-6). Eine weitere Einteilung in unterschiedliche Modellarten wird z.B. in [SZ03] vorgenommen. Hier wird zwischen Kontext-, Schnittstellen-, Schichten-, Kommunikations-, Daten-, Verhaltens- und objektbasierten Modellen unterschieden.

3. **Modellanalyse**
   Die Modellanalyse liefert Erkenntnisse über die Eigenschaften und das Verhalten des Systems. Hier werden beispielsweise die Finite-Elemente-Methode (FEM) oder die Mehrkörpersimulation (MKS) verwendet.

4. **Systemsynthese**
   Bei der Synthese werden die Simulations- und Berechnungsergebnisse der Modellanalyse auf das zu entwickelnde System übertragen und die Einstellung der Systemparameter optimiert.

5. **Systemanalyse**
   Das aus der Synthese entstandene System wird hier analysiert und bewertet. Weichen die Ergebnisse von den geforderten ab, so kann zu vorhergehenden Schritten zurückgesprungen werden. Gängige Verfahren sind unter anderem die Frequenzgangsanalyse oder die Analyse im Zeitbereich beispielsweise anhand von Springantworten.

Quelle: [VDI03]

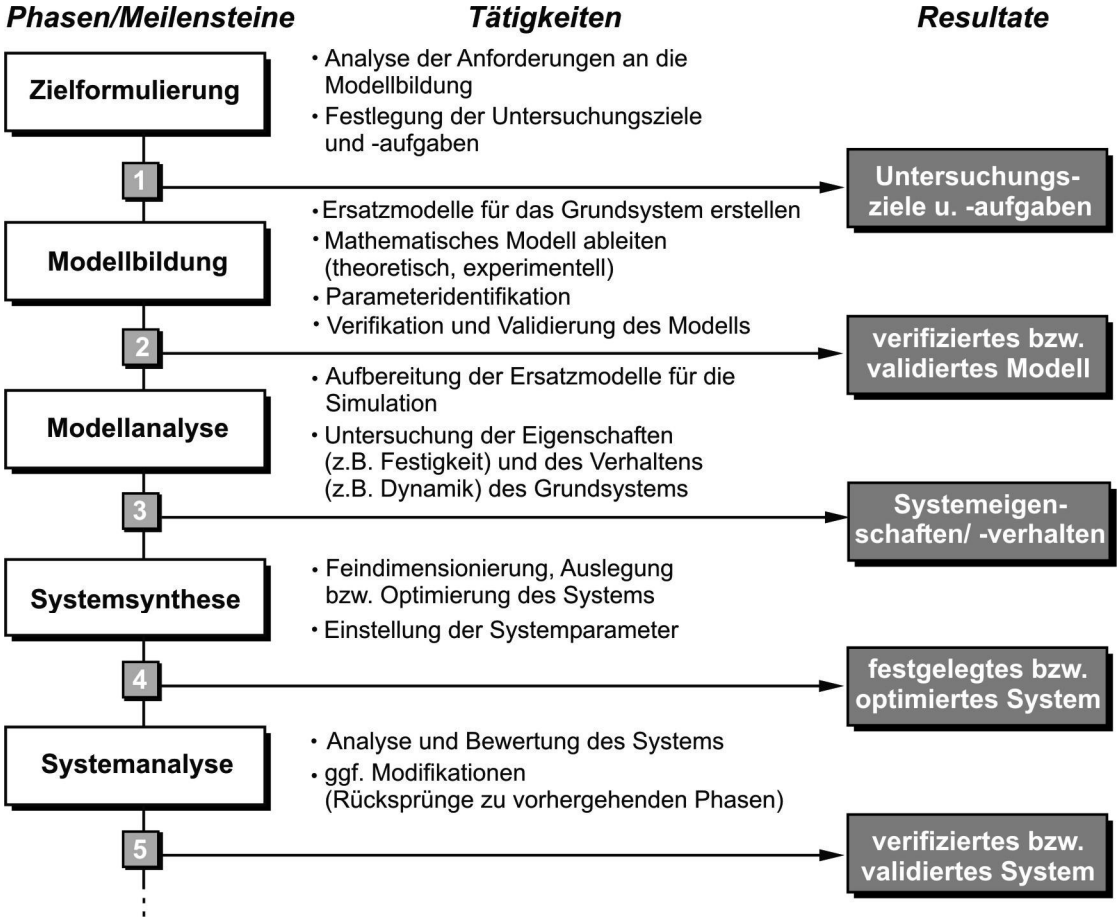


Abbildung 3-5: Vorgehen beim modellbasierten Entwickeln

Quelle: [VDI03]

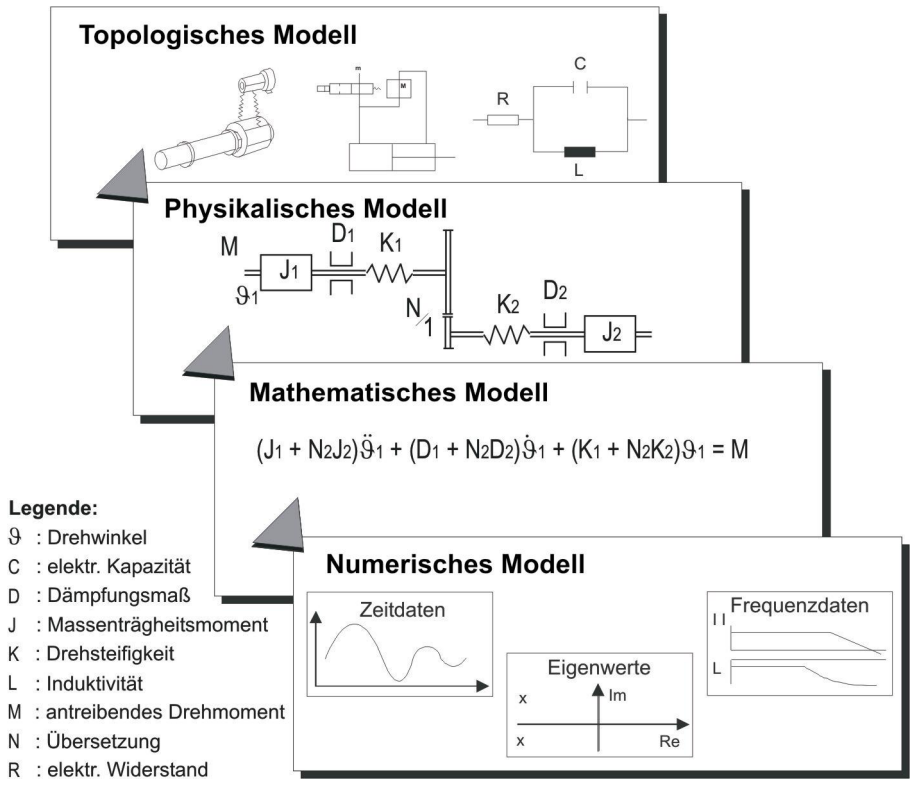


Abbildung 3-6: Abstraktionsebenen bei der Modellbildung

Mithilfe dieser Modelle können unterschiedliche Untersuchungen durchgeführt werden. Zusätzlich können daraus auch Vorgaben für elektrische oder geometrische Auslegungen gewonnen sowie Softwarealgorithmen generiert werden. In Abbildung 3-7 ist die Kopplung zwischen Modell und realem System am Beispiel eines Kraftfahrzeugs dargestellt. Abhängig von der durchzuführenden Untersuchung werden sowohl das zu entwickelnde System als auch die Umgebung Real oder als Modell benötigt.

1. **Funktionsnachweis**
   Hier liegen sowohl das zu entwickelnde System, als auch die Umgebung als Modell vor. Der Funktionsnachweis wird daher komplett mit virtuellen Komponenten erbracht. Diese Untersuchung wird auch als Software-in-the-Loop (SIL) bezeichnet.
2. **Applikation**
   Die Korrektheit der normalerweise am Streckenmodell überprüften Funktion kann hier am realen System nachgewiesen werden. Diese Untersuchung wird auch als Rapid Prototyping (RP) bezeichnet.
3. **Schnittstellennachweis**
   Durch die Kopplung des realen Steuergerätes mit dem virtuellen Streckenmodell kann die Fehlerfreiheit der Software und der Schnittstellen überprüft werden. Diese Untersuchung wird auch als Hardware-in-the-Loop (HIL) bezeichnet.
4. **Integration**
   Bei der Integration kann die Funktion am realen System in Betrieb genommen, überprüft und in einer Feinabstimmung optimal an die realen Verhältnisse angepasst werden.

Quelle: [BO01]

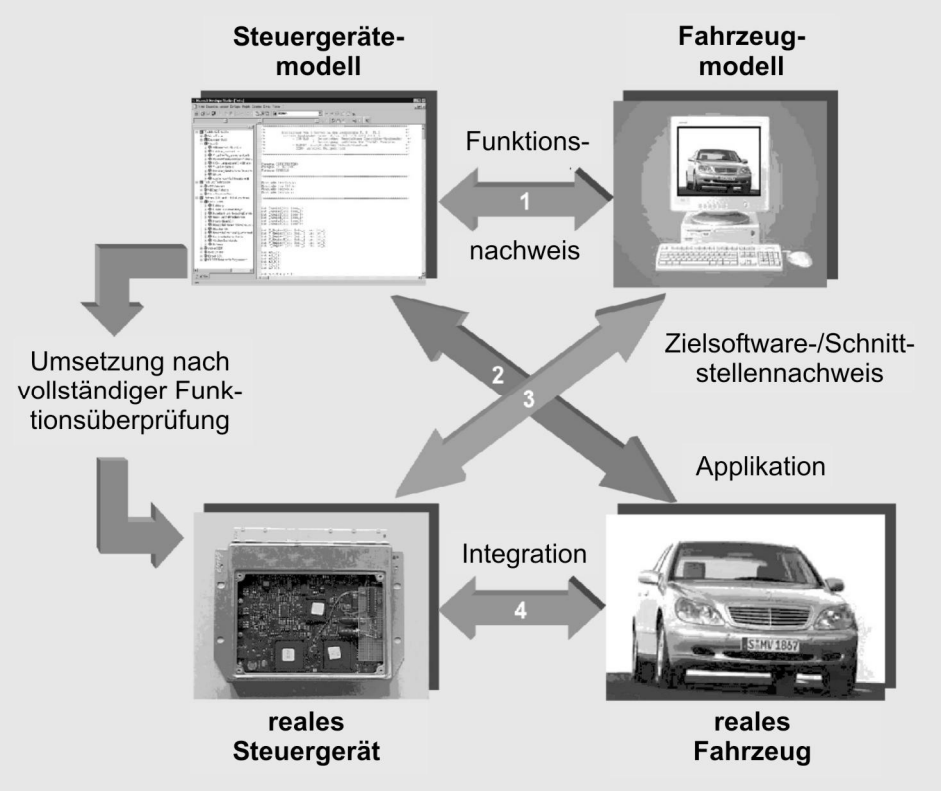


Abbildung 3-7: Kopplung zwischen Modell und realem System

3.3 Integration als Konstruktionsmethode

Als Integration soll nachfolgend der Zusammenschluss von Teilen (Funktionen, Strukturen, Elementen, Teilsystemen) zu einem übergeordneten Ganzen, dem System bzw. Produkt verstanden werden. Dieses System kann gegebenenfalls selbst wieder in ein übergeordnetes System integriert werden. In diesem Zusammenhang ist die Gestaltung der äußeren Schnittstellen von großer Bedeutung. In Abbildung 3-8 sind unterschiedliche Integrationsstufen am Beispiel eines Direktantriebes dargestellt [KZB+01].

Quelle: [KZB+01]

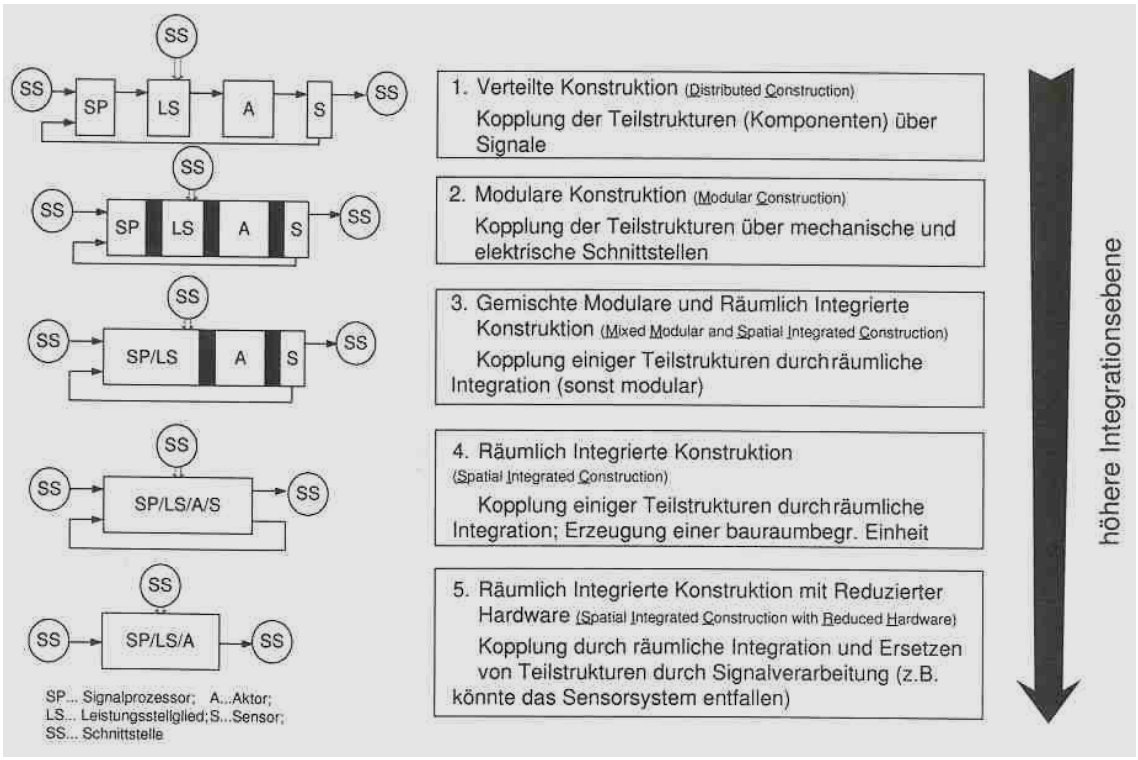


Abbildung 3-8: Integrationsstufen am Beispiel eines Direktantriebes

Die in Abbildung 3-8 dargestellten Integrationsstufen beziehen sich neben der Integration über Ein- und Ausgangsgrößen auch auf die räumliche Integration. Die Integration kann während eines Neuentwurfs als **Top-Down-Integration** bzw. während eines Redesigns als **Bottom-Up-Integration** durchgeführt werden. Der höchste Integrationsgrad kann bei der Top-Down Entwicklung erreicht werden, da die Anzahl der Einschränkungen geringer ist, als wenn man von vorgegebenen Subsystemen ausgeht.

3.4 Simultaneous Engineering

Unter Simultaneous Engineering versteht man die zielgerichtete, interdisziplinäre Zusammen- und Parallelarbeit aller Beteiligten während des gesamten Produktlebenszyklus. Dies betrifft nicht nur firmeninterne Organisationseinheiten sondern auch externe Beteiligte, wie beispielsweise Zulieferer [Küm99].

Bei der Fahrzeugindustrie existiert beispielsweise eine sehr ausgeprägte und definierte Arbeitsteilung zwischen dem Gesamtsystemhersteller und den Komponentenherstellern. Diese Beziehungen sind durch die Konzentration der Systemhersteller auf das Kerngeschäft und die Auslagerung der weniger zentralen Geschäftsfelder entstanden. Die Systemhersteller geben im Rahmen der Arbeitsteilung Grobkonzepte, Beschreibungen der Schnittstellen und Spezifikationen vor. Die Zulieferer setzen diese Vorgaben dann selbstständig in die entsprechenden Komponenten um [Koc00]. In Abbildung 3-9 ist dieses Zusammenspiel zwischen Systemhersteller und Zulieferer dargestellt. Im Allgemeinen hat jeder Systemhersteller mehrere Zulieferer, sowie die meisten Zulieferer wiederum für mehrere Hersteller arbeiten.

Quelle: [Koc00]

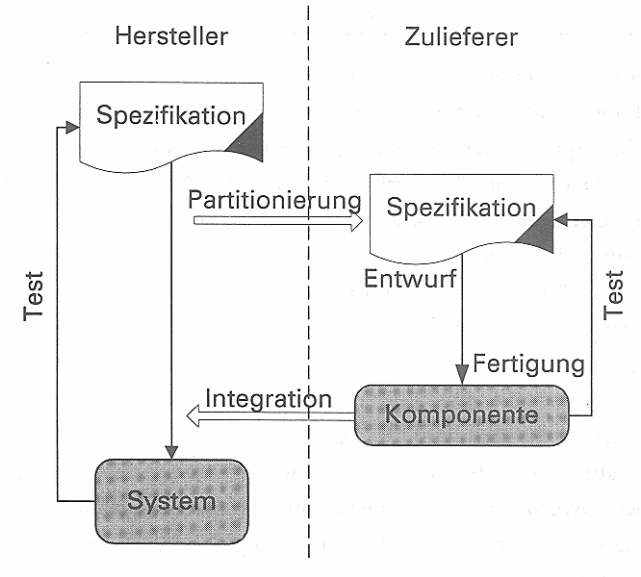


Abbildung 3-9: Simultaneous Engineering

3.5 Standards der Softwareentwicklung bei der Fahrzeugindustrie

3.5.1 Funktionsarchitektur nach CARTRONIC

Die Weiterentwicklung elektronischer Systeme im Kraftfahrzeug wird durch Forderungen nach wachsendem Funktionsumfang bei gleichzeitig anhaltendem Kostendruck bestimmt. Zur Lösung dieses Zielkonfliktes kann die Vernetzung der bisher weitgehend unabhängig voneinander arbeitenden Einzelsysteme zu einem fahrzeugweiten Verbund einen wesentlichen Beitrag leisten. Ein solches vernetztes System stellt hohe Anforderungen bezüglich Sicherheit, Zuverlässigkeit und Beherrschbarkeit durch den Fahrzeugnutzer. Um diese Kriterien bei stark gewachsener Komplexität erfüllen zu können, ist ein systematischer Aufbau des Systemverbunds unverzichtbar. Die Systematik muss zum einen geeignete Schnittstellen definieren und darüber hinaus das Zusammenwirken der Steuerungs- und Regelungssysteme koordinieren. Als Grundlage für eine Systemvernetzung im Kraftfahrzeug hat die Robert BOSCH GmbH eine solche Systematik unter dem Namen CARTRONIC entwickelt und bereits in Serienfahrzeugen zum Einsatz gebracht [BBR+98].

Die Grundprinzipien einer Funktionsarchitektur nach CARTRONIC sind dabei [JKM+02]:

- Jede Systemkomponente ist aus in sich abgeschlossenen Komponenten mit der minimal möglichen Anzahl von Schnittstellen aufgebaut.

- Jede Systemkomponente erfüllt autonom klar definierte Aufgaben, indem sie Informationen aufnimmt und Aufträge vergibt.

- Koordinatoren treffen Entscheidungen zur Koordination der Systemkomponenten.

- Aufträge werden hierarchisch vom Auftraggeber zum Steller verteilt.

- Die Schnittstellen jeder Systemkomponente sind so vielen anderen Komponenten wie nötig und so wenigen wie möglich bekannt.

In Abbildung 3-10 ist die Aufteilung des Fahrzeugs nach CARTRONIC in die vier Hauptfunktionseinheiten Antrieb, Fahrzeugbewegung, Karosserie und Innenraum sowie Bordnetz dargestellt. Als weitere Funktionseinheit ist Multimedia ebenfalls mit aufgenommen.

Quelle: [BHK00]

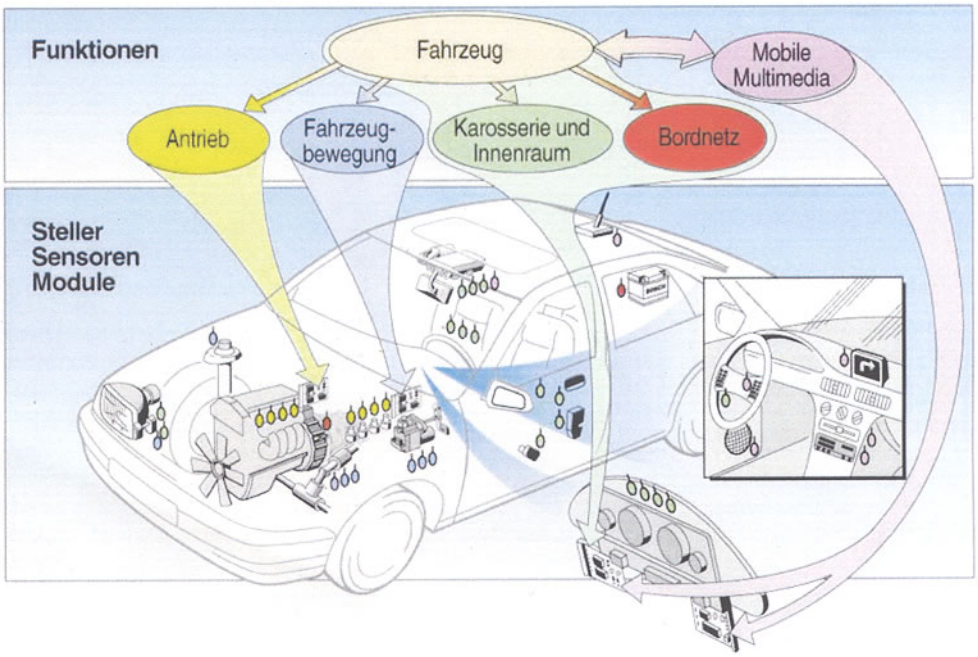


Abbildung 3-10: Funktionsarchitektur nach CARTRONIC

Der hierarchische Aufbau der vier Hauptfunktionseinheiten nach CARTRONIC kann der Abbildung 3-11 entnommen werden. In diesem Bild sind ebenfalls die Koordinatoren zu erkennen.

Quelle: [BBR+98], [JKM+02]

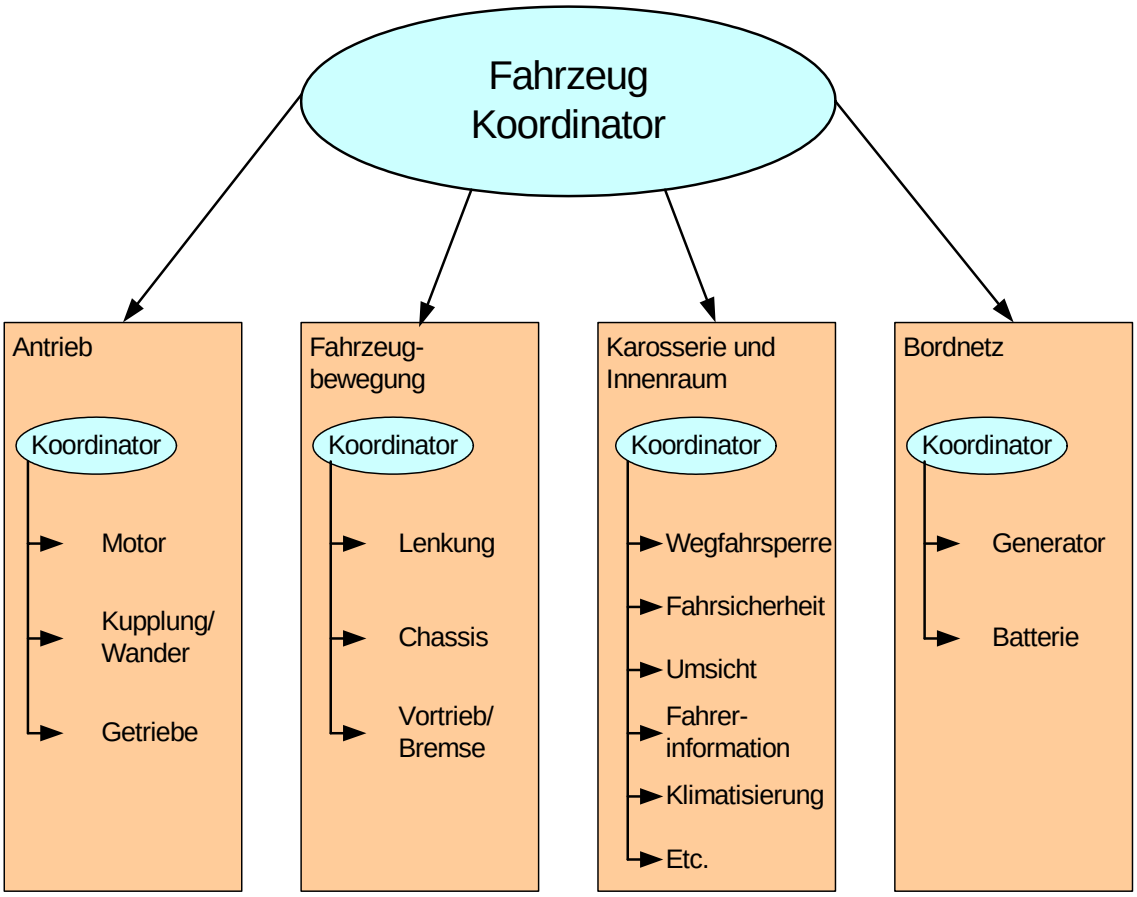


Abbildung 3-11: Hierarchische Strukturierung nach CARTRONIC

3.5.2 Betriebs- und Kommunikationssystem nach OSEK/VDX

OSEK/VDX ist ein von führenden Kfz-Herstellern, Zulieferern, Forschungseinrichtungen und Software-Herstellern ins Leben gerufener Standard für ein modulares, statisches Echtzeit-Betriebssystem. Bereits 1993 haben sich dazu BMW, Bosch, DaimlerChrysler, Opel, Siemens, VW und die Universität Karlsruhe zum OSEK Arbeitskreis zusammengeschlossen. Die Abkürzung OSEK steht für "**O**ffene **S**ysteme und deren Schnittstellen für die **E**lektronik im **K**raftfahrzeug", VDX für das französische Projekt "**V**ehicle **D**istributed e**X**ecutive" von Renault und PSA. Beide wurden 1994 zu dem Gemeinschaftsprojekt OSEK/VDX zusammengeführt. Mittlerweile gehören dem OSEK/VDX Arbeitskreis über 60 Unternehmen an. Beide Titel beschreiben bereits als vorrangiges Einsatzfeld die elektronischen Systeme in Automobilen. Der Standard ist völlig plattformunabhängig definiert und stellt nur vergleichsweise geringe Anforderungen an das Zielsystem. Der Grund hierfür liegt darin, dass die Steuerung vieler Komponenten in Kraftfahrzeugen von sehr einfachen und kostengünstigen Prozessoren übernommen werden muss. Nach oben sind der Zielplattform praktisch keine Grenzen gesetzt, so dass auch komplexe CPUs sinnvoll genutzt werden können.
Der Standard definiert ursprünglich drei Komponenten als eigenständige Bestandteile von OSEK/VDX:

- das eigentliche Betriebssystem (OSEK-OS),
- das Kommunikations-Subsystem (OSEK-COM)
- und das Netzwerk-Management (OSEK-NM).

Ein Vorteil der getrennten Definition liegt darin, dass so Revisionen einzelner Komponenten wesentlich vereinfacht werden.

Mit OSEK/VDX soll die Portierbarkeit und die Wiederverwendbarkeit von Steuerungssoftware durch folgende Punkte erreicht werden:

- Spezifikation von abstrakten, anwendungsunabhängigen Schnittstellen für die Gebiete: Echtzeit-Betriebssystem, Kommunikation, Netzwerkmanagement,
- Spezifikation von hardwareunabhängigen Benutzerschnittstellen,
- effizientes Design: die Funktionalität muss konfigurierbar und skalierbar sein, damit eine optimale Anpassung an die Bedürfnisse der Anwendung erfolgen kann.

Die Umsetzung dieser Punkte spart Entwicklungszeit und -kosten, erhöht die Qualität der Software und verbessert die Unabhängigkeit der einzelnen Implementierungen. Für unterschiedliche Architekturen von Steuergeräten stehen einheitliche Schnittstellen zur Verfügung. Durch OSEK/VDX wird die Steuerung der verteilten Ressourcen in einem Kraftfahrzeug geregelt. Es erhöht dadurch die Leistung des Gesamtsystems ohne zusätzliche Hardware erforderlich zu machen. Steuerungen im Fahrzeugbau sind durch hohe Echtzeitanforderungen gekennzeichnet. Deshalb bietet das Betriebssystem OSEK-OS die nötige Funktionalität, um ereignis- und zeitgesteuerten Systemen gerecht zu werden. Die spezifizierten Funktionen des Betriebssystems bilden eine Basis, um verschiedene Software-Module von unterschiedlichen Herstellern einfügen zu können. OSEK/VDX bietet so die Möglichkeit, die speziellen Anforderungen individueller Steuerungseinheiten zu berücksichtigen. Außerdem lag bei der Realisierung von OSEK/VDX das Hauptaugenmerk nicht auf vollständiger Kompatibilität, sondern auf der einfachen Portierungsmöglichkeit der Softwarekomponenten. Da das Betriebssystem für sämtliche Typen von Steuergeräten eingesetzt werden soll, muss es eine Fülle von Anwendungen auf vielen verschiedenen Hardwarearchitekturen ermöglichen. Seine Modularität und Konfigurationsmöglichkeiten machen es sowohl an low-end Mikrocontroller, als auch an komplexe Steuergeräte anpassbar. Hierzu dient die Definition von so genannten Konformitätsklassen, die Auswahl der Scheduling Strategie und die anwendungsspezifische Anpassbarkeit verschiedener Systemparameter. Aus Sicherheitsgründen, zur Garantie der Echtzeiteigenschaften und wegen der erheblich besseren Optimierungsmöglichkeiten ermöglicht OSEK/VDX keine dynamische Erzeugung von Systemobjekten. Es muss bereits bei der Generierung des Kernels die genaue Anzahl der benötigten Betriebsmittel feststehen. Diese kann zur Laufzeit auch nicht mehr verändert werden. Auch eine dynamische Speicherverwaltung ist nicht vorgesehen. [OSEK], [SBR03], [SZ03]

Sämtliche Dokumente des Standards, z.B. die API-Beschreibung für OSEK-OS, OSEK-COM und OSEK-NM sind frei verfügbar [OSEK].

3.5.3 Mess- und Automatisierungsschnittstellen nach ASAM

Der **A**rbeitskreis zur **S**tandardisierung von **A**utomatisierungs- und **M**eßsystemen (ASAM) mit den Arbeitsgruppen **M**easurement, **C**alibration und **D**iagnostics (MCD) wurde 1991 durch die deutschen Automobilhersteller Audi, BMW, Mercedes Benz (DaimlerChrysler), Porsche und VW gegründet. Angetrieben durch den Wunsch, den Aufwand für Erstellung, Pflege und Wartung von Mess- und Automatisierungssystemen zu reduzieren, wurden folgende Ziele definiert:

- Modularisierung von Mess- und Automatisierungssystemen und
- Definition von Standard-Schnittstellen.

Inzwischen besteht der ASAM aus über 80 Mitgliedsfirmen. In Abbildung 3-12 sind die Schnittstellenstandards ASAM-MCD 1b, 2 und 3 dargestellt.

Der ASAM-MCD 1b Standard beschreibt die Schnittstelle zur Einbindung entsprechender Geräte in Mess-, Applikations- und Diagnosesysteme.

Der ASAM-MCD 2 Standard beschreibt Mess- und Verstellgrößen bezüglich Speicherbereich, Umrechnungsformeln und Zugriffsmechanismen durch Mess-, Applikations- und Diagnosesysteme.

Der ASAM-MCD 3 Standard beschreibt die Automatisierungs- und Fernsteuerschnittstellen von Mess-, Applikations- und Diagnosesystemen.

Quelle: [SZ03]

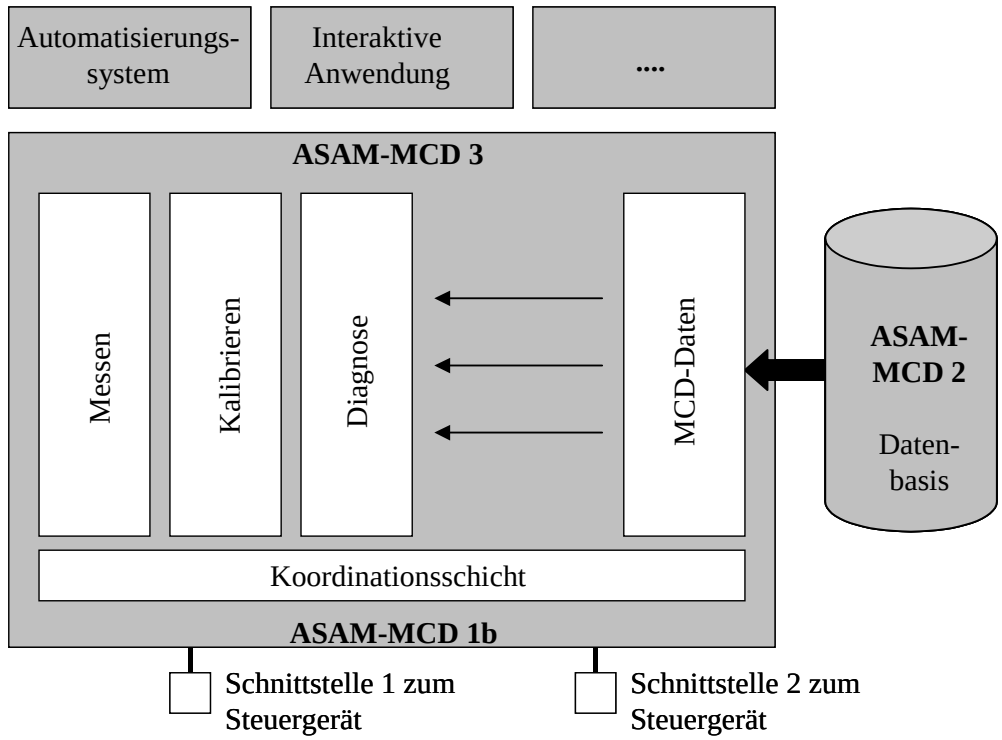


Abbildung 3-12: ASAM Standards

Kapitel 4

ANALYSE UND BEWERTUNG VON RAPID PROTOTYPING ALS ENTWURFSMETHODE FÜR EINGEBETTETE SYSTEME

4 Analyse und Bewertung von Rapid Prototyping als Entwurfsmethode für eingebettete Systeme

Wie unter anderem in den vorangegangenen Abschnitten dargestellt, werden sich die meisten Innovationen in naher Zukunft vorwiegend durch Fortschritte im Bereich Elektrotechnik und Informationstechnik realisieren lassen. Die weiteren Betrachtungen werden sich daher auf diese Technologien fokussieren. Nicht betrachtet werden soll hingegen das weit verbreitete Rapid Prototyping im Bereich Maschinenbau, welches für die schnelle Fertigung von dreidimensionalen Körpern (z.B. mittels Stereolithographie) zur Überprüfung von Design, Statik, Passgenauigkeit, etc. eingesetzt wird.

Aufgabe des Rapid Prototyping (schnelle Systemprototypenentwicklung) ist es, die spezifizierten Funktionen möglichst schnell und automatisch in eine Realisierung zu überführen, damit die Funktion in einer realen Systemumgebung erlebbar wird. Rapid Prototyping dient also dazu, so früh wie möglich umfassende Aussagen über die gewünschte Funktionalität und die Erfüllung z.B. zeitlicher Bedingungen zu gewinnen. Zudem soll das Rapid Prototyping relevante Anforderungen und Probleme aufzeigen sowie als Diskussionsbasis zwischen allen Beteiligten dienen und damit eine Entscheidungsfindung beschleunigen und absichern. Ein weiterer wichtiger Punkt ist die Gewinnung praktischer Erfahrungen in ersten Experimenten. Dies ist insbesondere dann notwendig, wenn die Erstellung eines Modells der realen Umgebung für eine Simulation sehr aufwändig ist oder wenn Aufgaben zu erfüllen sind, welche mit dem menschlichen Wahrnehmungsvermögen zu tun haben. Dies gilt z.B. für angenehme Schaltvorgänge von Automatengetrieben sowie die Bewertung von Motorgeräuschen oder für Algorithmen aus dem Bereich der Audio- und Videodatenverarbeitung. [MBS+00]

Die Prototypenentwicklung kann prinzipiell in zwei verschiedene Zielrichtungen unterteilt werden (Abbildung 4-1):

- **Horizontale Prototypen**
  zielen auf die Darstellung eines breiten Bereichs des Systems, stellen aber eine abstrakte Sicht dar und vernachlässigen Details.
- **Vertikale Prototypen**
  stellen einen eingeschränkten Bereich des Systems recht detailliert dar.

Weitergehende Informationen sind z.B. in [SZ03] zu finden.

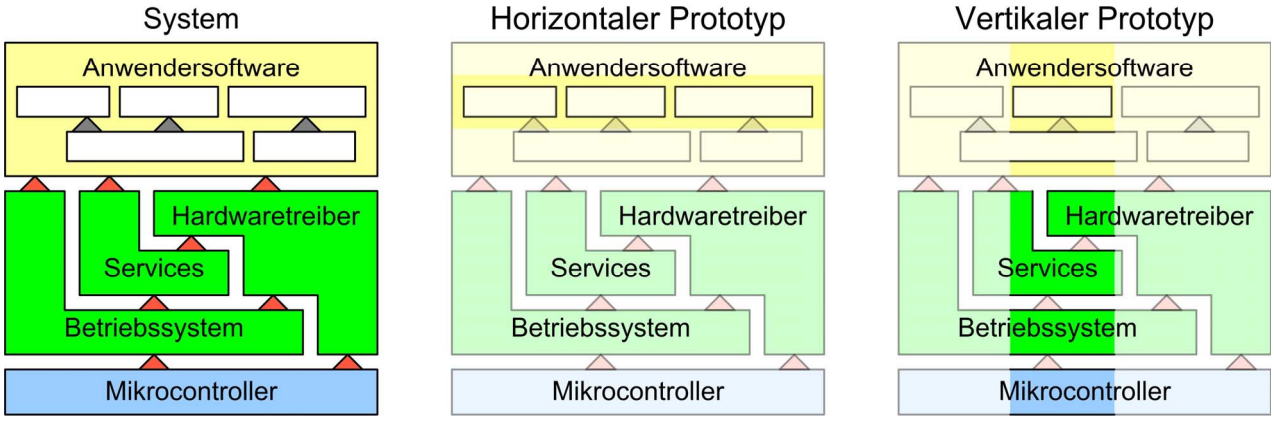


Abbildung 4-1: Horizontale und vertikale Prototypen

Im Folgenden wird auf die, bei der Fahrzeugindustrie zur Entwicklung elektronischer Steuer- bzw. Regelungen, bereits eingeführten Entwicklungsmethoden des Rapid Prototyping wie das Arbeiten mit einem **„Universal Experimentiersystem“** und das Arbeiten im **„Bypass“** eingegangen. Dazu wird verdeutlicht, was im Bezug auf diese Arbeit unter diesen Begriffen zu verstehen ist. Dabei wird unter anderem auch auf den **„externen Bypass“** und den **„internen Bypass“** kurz eingegangen. Es werden die Stärken und Einschränkungen dieser teilweise sehr unterschiedlichen Ansätze dargestellt und bewertet. Dies wird unter anderem anhand einer Einordnung der Flexibilität und Performance versus der Identität mit dem angestrebten Zielsystem vorgenommen. Zudem wird dargestellt, von wem welcher Aufwand bei den unterschiedlichen Ansätzen erbracht werden muss.

Anhand dieser Analyse wird aufgezeigt, welcher der vorgestellten Ansätze für welche Aufgabenstellung am besten geeignet ist und in welchem Bereich es noch Defizite und somit Möglichkeiten für Optimierungen gibt.

4.1 Universal Rapid Prototyping Experimentiersystem

Unter dem Sammelbegriff **„Universal Rapid Prototyping Experimentiersystem“** sind jene Systeme zusammengefasst, welche das Zielsystem mit Standardkomponenten nachbilden. Die Basis dieser Systeme ist häufig ein Standard-Bussystem wie z.B. VME oder PCI. Bei diesen Systemen ist es relativ schnell möglich, einen Prototyp aufzubauen, da viele I/O-Interfaces als Standard-Karten erhältlich sind. Die I/O-Interfaces können als passive Karten (ohne Prozessor) oder als aktive Karten (mit Prozessor) ausgeführt sein. Zudem ist es sehr oft möglich, die benötigte Rechenleistung durch unterschiedliche Prozessorkarten zu skalieren. Die Verwendung einer grafischen Entwicklungsumgebung ermöglicht es für jede Aufgabe die am besten passende Beschreibungsform zu verwenden. Die Berechnungen werden üblicherweise in Floating Point Arithmetik durchgeführt. Wie in Abbildung 4-2 dargestellt, werden sämtliche für den entsprechenden Prototypen benötigten Karten in einem Standardgehäuse untergebracht, welches die entsprechende Busbackplane und Netzteile enthält.

Quelle: [Kas02]

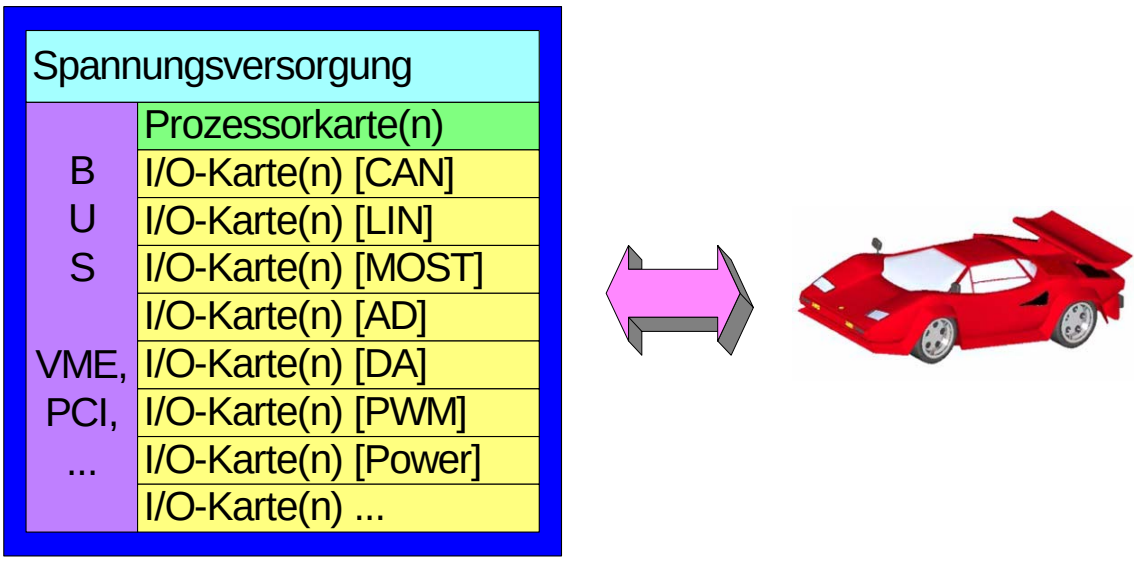


Abbildung 4-2: Universelles Experimentiersystem

Dieser sehr flexible Ansatz führt aber häufig zu einer vergleichsweise großen Bauform. Zudem weisen die für den allgemeinen und breiten Einsatz vorgesehenen I/O-Interfaces große Unterschiede zu denen des Zielsystems auf. Das gilt sowohl für die Hardware, als auch für die Software. Wenn auch noch unterschiedliche Betriebssysteme zum Einsatz

kommen, können die mit dem Universal Experimentiersystem erzielten Ergebnisse nur sehr aufwändig auf das spätere Zielsystem portiert werden.

Dieses Vorgehen wird daher häufig für die Realisierung von horizontalen Prototypen verwendet.

Bewertung „Universal Rapid Prototyping Experimentiersystem“

Vorteile:

- Leichte Erweiterbarkeit
- Große Auswahl an I/O-Interfaces
- Skalierbare Rechenleistung
- Schnelle Konfigurierbarkeit

Nachteile:

- Große Bauform
- Externe Signalkonditionierung erforderlich
- I/O-Hardware nicht identisch mit Zielsystem
- I/O-Softwaretreiber meist sehr unterschiedlich zum Zielsystem
- Betriebssystem häufig unterschiedlich zum Zielsystem
- Schlechte oder aufwändige Weiterverwendbarkeit der Ergebnisse
- Typischerweise nur bedingt Fahrzeugtauglich

4.2 Bypass

Von einem „Bypass“ spricht man, wenn als Prozess- bzw. I/O-Schnittstelle ein Seriensteuergerät verwendet wird, welches sowohl Software- als auch Hardwaretreiber aus der Serie beinhaltet. Zudem muss dem Anwender über einen so genannten Freischnitt ermöglicht werden, bestehende Funktionen zu modifizieren bzw. neue Funktionalität zu integrieren. Der Freischnitt ermöglicht es der Bypass-Software, die von ihr benötigten Eingangsgrößen auszulesen und die von ihr berechneten Ausgangsgrößen wieder in die Seriensoftware einfließen zu lassen. Die Verwendung einer grafischen Entwicklungsumgebung ermöglicht es für jede Aufgabe die am besten passende Beschreibungsform zu verwenden. Die Berechnungen in der Bypass-Software werden üblicherweise in Floating Point Arithmetik durchgeführt, wohingegen die Berechnungen in der „Serien-Software überwiegend in Integer Arithmetik durchgeführt werden. In Abbildung 4-3 ist eine „Serien-Software mit drei zeitsynchronen und einem kurbelwellenwinkelsynchronen Rechenraster dargestellt. Funktionen aus drei dieser Raster wurden freigeschnitten.

Quelle: [Kas02]

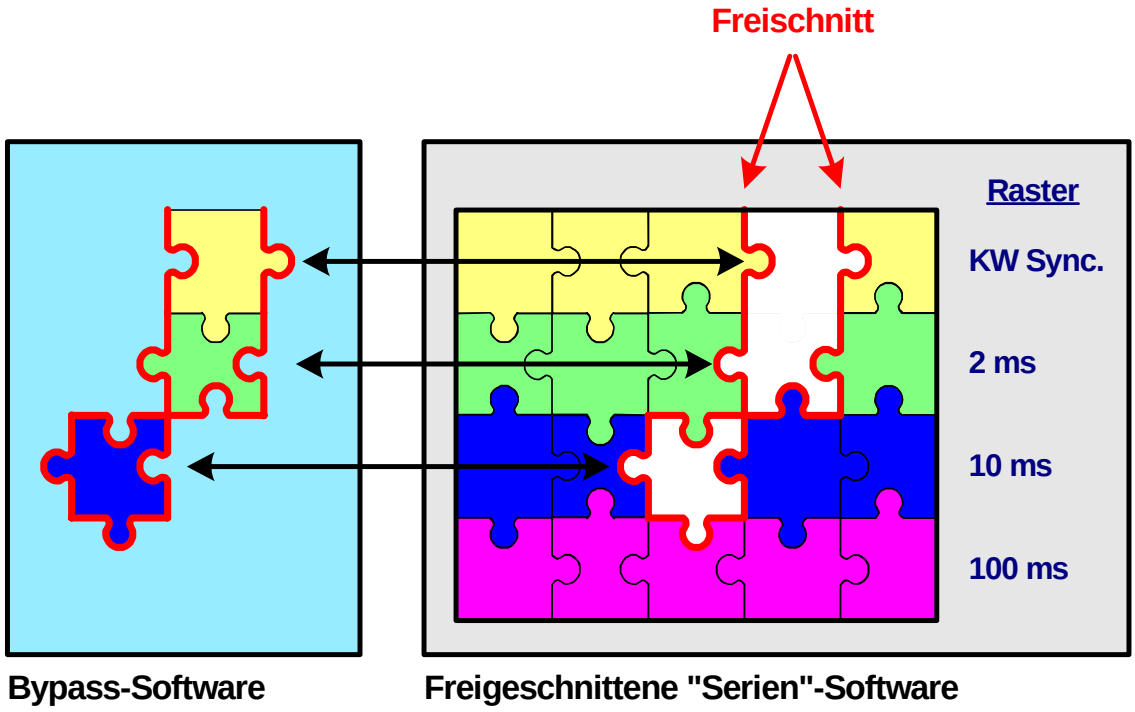


Abbildung 4-3: Freischnitt

Der Bypass allgemein kann weiterhin in zwei Realisierungsformen unterteilt werden. Zum einen in den „externen Bypass“ sowie zum anderen in den „internen Bypass“. Im Folgenden werden die beiden Varianten kurz vorgestellt.

4.2.1 Externer Bypass

Beim „externen Bypass“ werden die Teilfunktionalitäten, welche modifiziert oder neu hinzugefügt werden sollen, über eine externe physikalische Schnittstelle (z.B. CAN [Ets02], Ethernet, Fire Wire, etc.) an einen oder mehrere Rechenknoten ausgelagert (Abbildung 4-4). Eine sehr schnelle und kompakte Lösung, welche den Bypassrechner von Kommunikationsaufgaben weitgehend entlastet, ist der Emulatortastkopf (ETK). Weitere Informationen über die ETK Schnittstelle sind in [Wen04] zu finden. Ist der I/O-Umfang, der vom Seriensteuergerät (Bypassrechner) zur Verfügung gestellt wird nicht ausreichend, kann das Bypass-System ebenfalls noch weitere I/O-Schnittstellen zur Verfügung stellen.

Quelle: [Kas02]

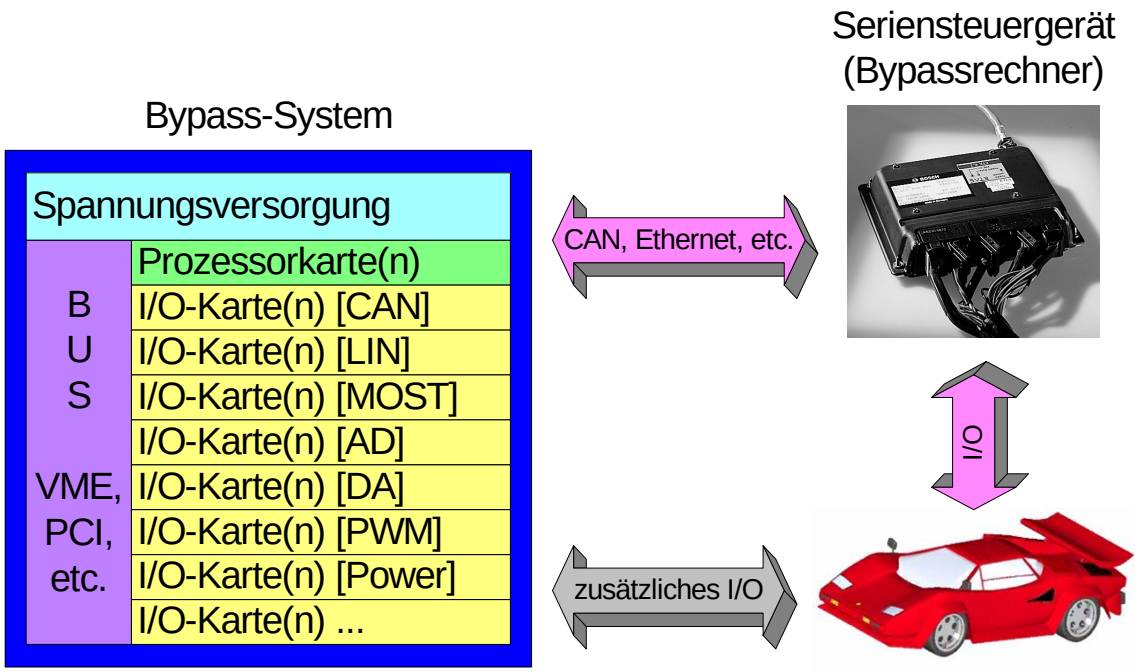


Abbildung 4-4: Externer Bypass

In den folgenden Abbildungen ist exemplarisch die zeitliche Abfolge von fünf Prozessen in einer Task dargestellt. Es sind jeweils zwei aufeinander folgende Taskaktivierungen zu sehen. Weitere Informationen zu Prozessen und Tasks können der Literatur [Eta02_1], [Eta02_2], [Eta02_3] entnommen werden. Es soll beispielsweise Prozess P2 im Bypass berechnet werden (Abbildung 4-5).

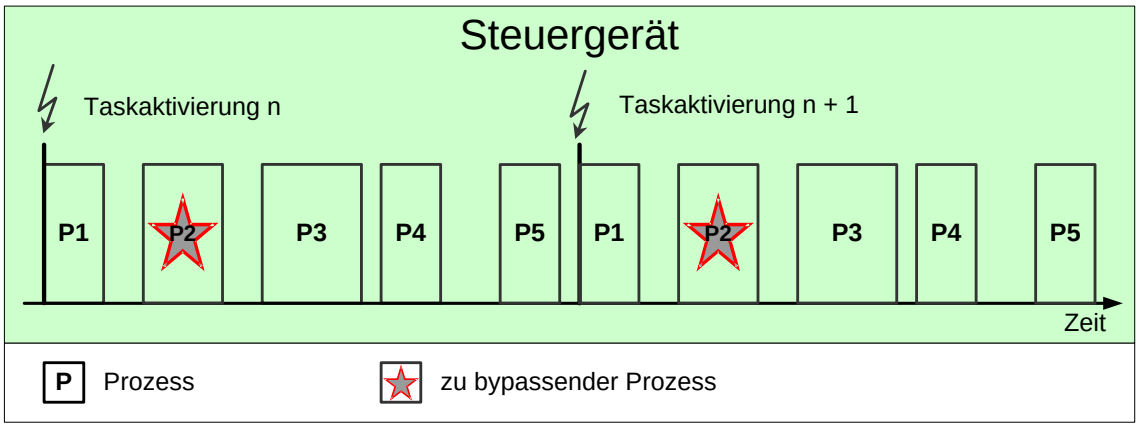


Abbildung 4-5: Festlegung der Bypassfunktion

Die für den Bypass benötigten Eingangsgrößen müssen vom Seriensteuergerät über die Bypass-Schnittstelle zum Bypass-System kommuniziert werden. Die Verfügbarkeit neuer Daten wird dem Bypass-System durch einen Interrupt mitgeteilt, welcher auch die Taskaktivierung auslöst (Abbildung 4-6).

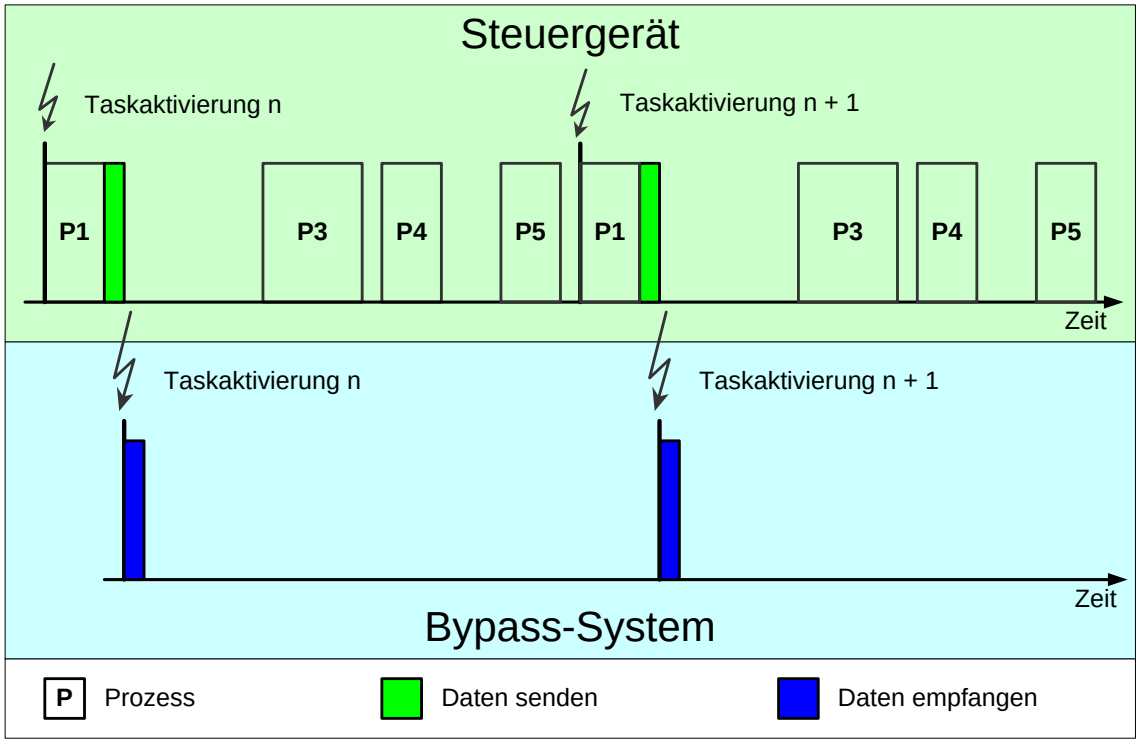


Abbildung 4-6: Bypass-Timing Daten lesen

Das Bypass-System berechnet daraus dann die entsprechenden Ausgangsgrößen (Abbildung 4-7).

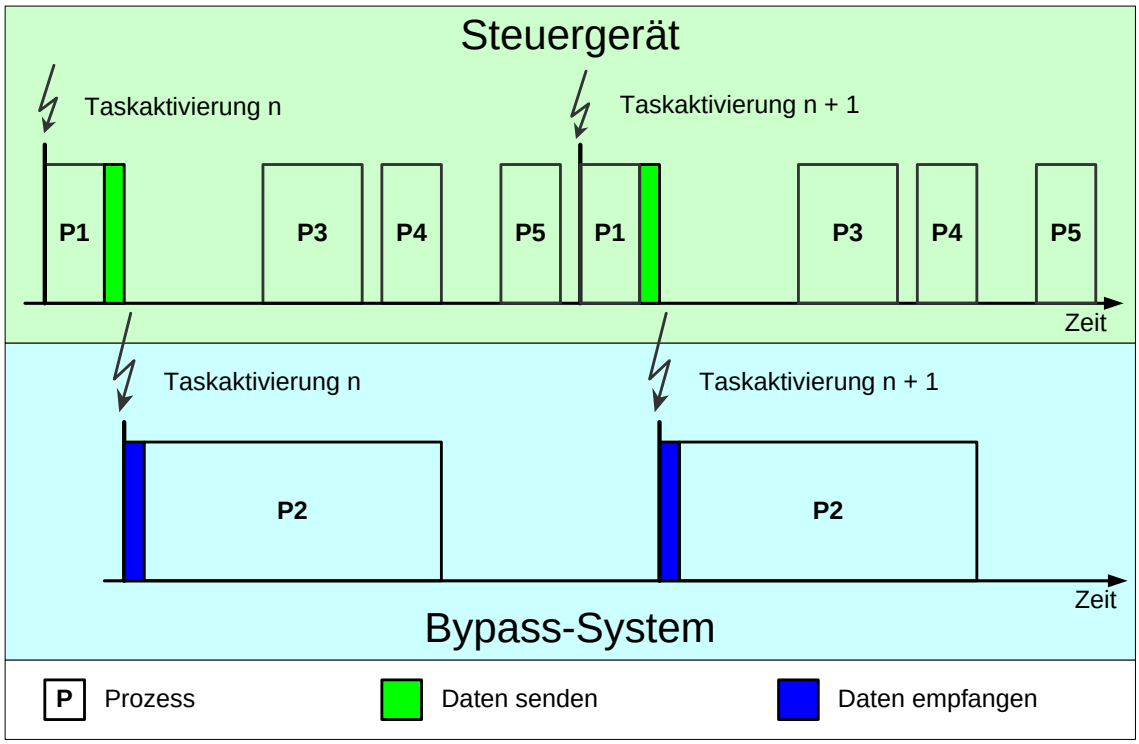


Abbildung 4-7: Bypass-Timing berechnen

Die berechneten Ausgangsgrößen werden dann wiederum über die Bypass-Schnittstelle zum Seriensteuergerät übertragen. Das Seriensteuergerät nimmt diese Werte für die weitere Verarbeitung entgegen (Abbildung 4-8).

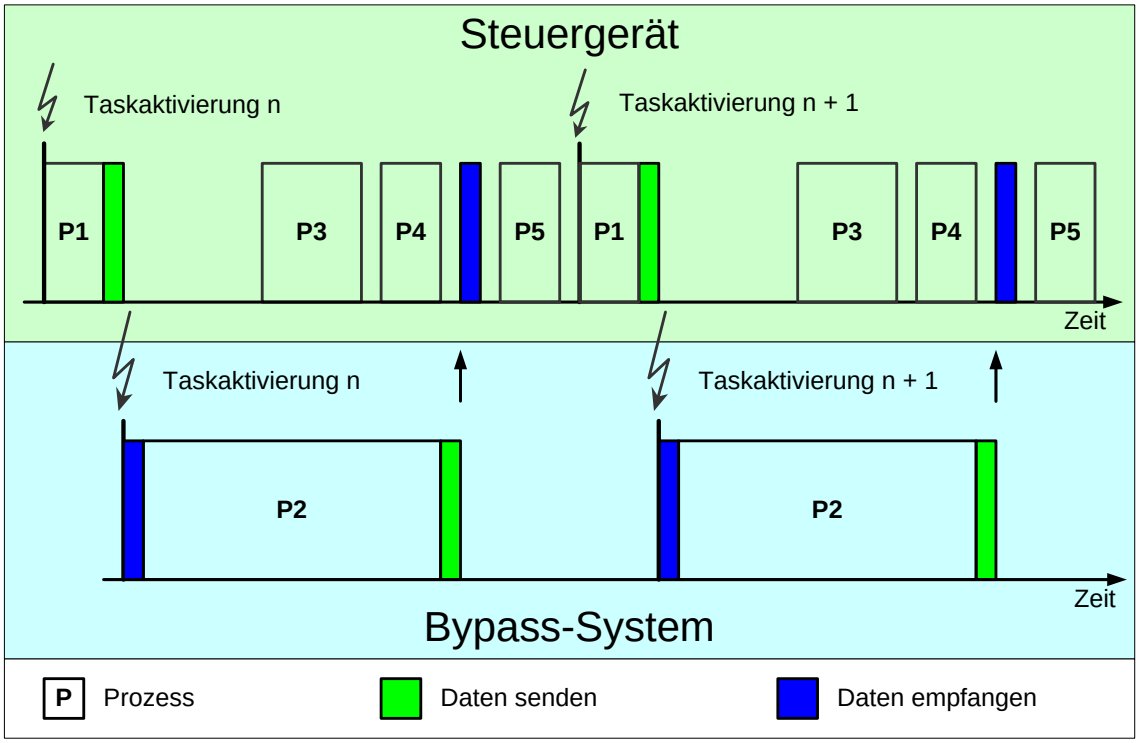


Abbildung 4-8: Bypass-Timing Daten schreiben

Da die Kommunikation und Berechnung mit Laufzeiten behaftet sind, kann es speziell bei Bypässen von schnellen Rechenrastern vorkommen, dass das Ergebnis erst nach der nächsten Taskaktivierung zur Verfügung steht. Das Seriensteuergerät bekommt dann Bypassdaten, welche mindestens eine Taskaktivierung zurück liegen. Dieser Sachverhalt wird daher auch als Rasterverzug bezeichnet (Abbildung 4-9).

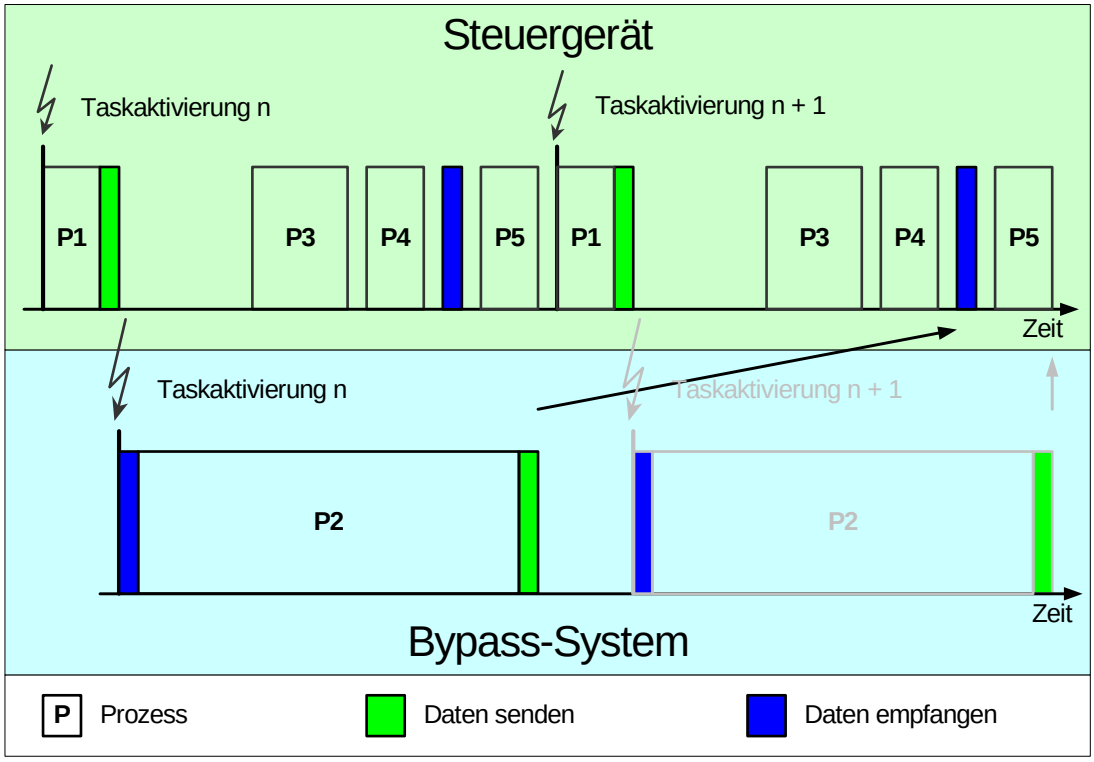


Abbildung 4-9: Bypass-Timing mit Rasterverzug

Zum Datenaustausch zwischen dem Seriensteuergerät und dem Bypass-System, wird häufig das DISTAB-Verfahren verwendet. Mit diesem Verfahren ist es möglich interne Größen des Steuergerätes zu lesen und zu verändern. Die Adressen der Größen, welche aus dem Steuergerät gelesen werden sollen, werden nicht hart codiert, sondern über eine von extern beschreibbare Tabelle vorgegeben. Damit ist es möglich, beliebige Größen auszulesen ohne den Softwarestand des Steuergerätes zu ändern. Detaillierte Informationen zur Arbeitsweise des DISTAB- Verfahrens sind in [Wen04] zu finden.

In vielen Fällen wird der Freischnitt vom Systemlieferant in einer standardisierten Form realisiert. Bei dieser standardisierten Form werden die entsprechenden Werte für das Bypass-System ausschließlich am Taskende kommuniziert. Um die Werte des Bypass-Systems wieder ins Steuergerät einfließen zu lassen sind an einigen vom Systemlieferant festgelegten Stellen Umschaltpunkte realisiert. An diesen Umschaltstellen kann mit Hilfe eines Applikationswerkzeugs festgelegt werden, ob die Werte vom Bypass-System oder vom Steuergerät verwendet werden sollen. Diese Form des Freischnittes reduziert zwar den Aufwand für den Systemlieferant, schränkt aber den Einsatz nicht unerheblich ein. Unter anderem ist ein Bypass ohne Rasterverzug mit dieser Form des Freischnittes prinzipiell nicht möglich.

Das Verfahren des „externen Bypass“ hat, wenn man nur den Teil des Seriensteuergerätes betrachtet, eine relativ kleine Bauform und einen großen Anteil zielsystemidentischer Komponenten der I/O-Hardware und Software. Da das Betriebssystem des Seriensteuergerätes ebenfalls sehr oft identisch mit dem des Zielsystems ist, können die erzielten Ergebnisse relativ einfach auf dem zukünftigen Zielsystem weiterverwendet werden. Bezieht man das Bypass-System in die Betrachtung mit ein, so kommen als weitere Vorteile noch die relativ einfache Erweiterbarkeit des I/O-Umfangs sowie die Skalierbarkeit der Rechenleistung hinzu.

Das führt natürlich zwangsläufig dazu, dass die Bauform über das gesamte System betrachtet relativ groß ausfällt. Zudem wird eine entsprechend leistungsfähige Schnittstelle zwischen dem Seriensteuergerät und dem Bypass-System benötigt. Die Performance dieser Bypass-Schnittstelle trägt einen wesentlichen Anteil zur Leistungsfähigkeit des Gesamtsystems bei. Um diese Schnittstelle vom Seriensteuergerät bedienen zu können, ist ein „Freischnitt“ erforderlich. Dieser kann in der Regel nur vom Steuergerätehersteller selbst bzw. von jemandem mit Zugriff auf dessen Entwicklungsprozess vorgenommen werden. Der Kunde ist daher immer darauf angewiesen, dass jemand für ihn diesen Freischnitt durchführt. Die Implementierung dieses „Freischnittes“ ist zudem wesentlich dafür verantwortlich, ob ein Bypass ohne Rasterverzug überhaupt möglich ist. Verwendet man beispielsweise wie in Abbildung 4-10 dargestellt standardisierte Verfahren mit Datenübertragung am Ende des Rechenrasters, so ist nur ein Bypass mit Rasterverzug möglich. Da die hardwarenahe Software auf dem Seriensteuergerät verbleibt, sind Änderungen in dieser nicht möglich. Zudem können Modifikationen an der Hardware selbst nur mit erheblichem Aufwand vorgenommen werden. Bezieht man die I/O-Interfaces des Bypass-Systems mit ein, erhält man damit erneut eine schlechte Portierbarkeit auf das spätere Zielsystem.

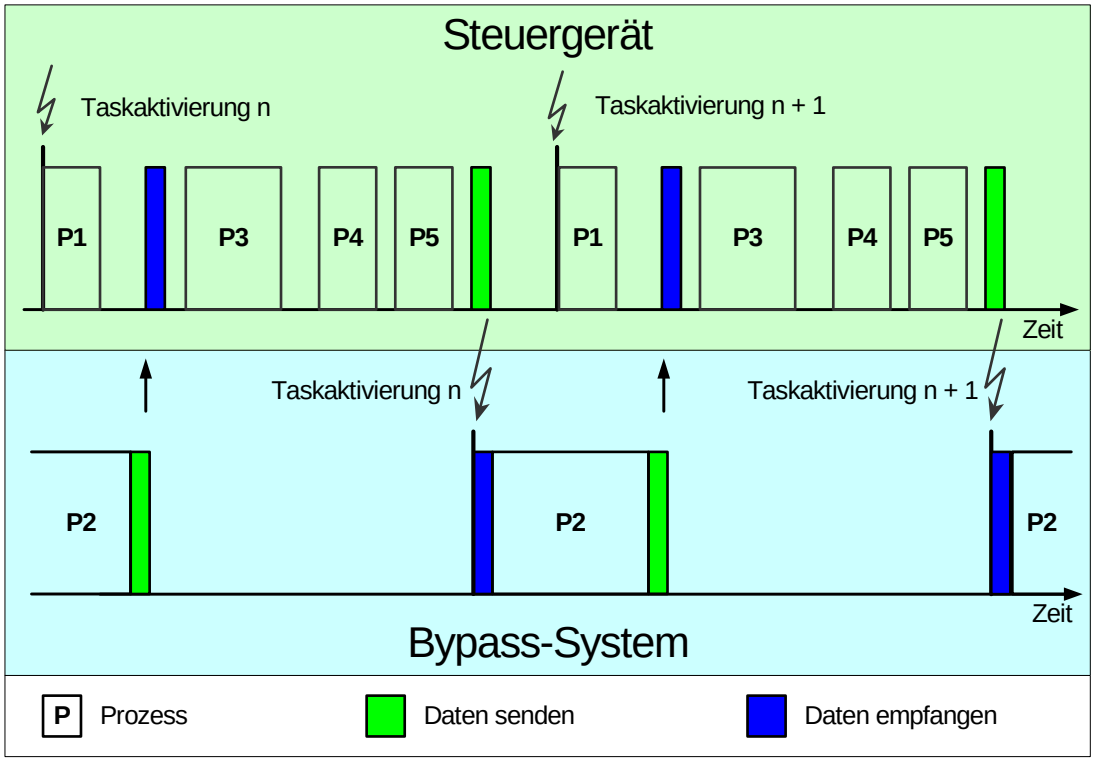


Abbildung 4-10: Bypass-Timing mit Datenübertragung am Ende des Rechenrasters

Bewertung "Externer Bypass"

Vorteile:

- Kleine Bauform des Bypassrechners
- I/O von Bypassrechner meist identisch mit Zielsystem
- Leichte Erweiterbarkeit der I/Os des Entwicklungssystems
- I/O-Softwaretreiber des Bypassrechners sehr ähnlich oder identisch mit Zielsystem
- Betriebssystem des Bypassrechners meist identisch mit Zielsystem
- Einfache Weiterverwendbarkeit der Ergebnisse
- HW-Schnittstelle (z.B. ETK) bei Applikationssteuergerät bereits vorhanden
- Funktionsentwicklung auch ohne Steuergeräte-Entwicklungsumgebung

Nachteile:

- Entwicklungssystem erforderlich
- Große Bauform des Gesamtsystems
- Schnittstelle zwischen Entwicklungssystem und Bypassrechner erforderlich
- Eingeschränkte Performance der Schnittstelle zwischen Entwicklungssystem und Bypassrechner führt bei Standard-Implementierung zu Rasterverzug
- Durch Rasterverzug nicht alle Funktionen im Bypass darstellbar
- Zusätzliches I/O von Entwicklungssystem wenig identisch mit Zielsystem
- Freischnitt in der Software des Bypassrechners erforderlich

4.2.2 Interner Bypass

Der „interne Bypass“ ist prinzipiell vergleichbar mit dem „externen Bypass“. Der wesentliche Unterschied besteht darin, dass die Daten nicht über eine Schnittstelle aus dem Steuergerät heraus zu einem externen System gesendet, sondern intern vom Mikrocontroller verarbeitet werden.

Bei diesem Ansatz ist die Bauform verglichen mit den zuvor beschriebenen Ansätzen am kleinsten, da neben dem Seriensteuergerät keine weitere Hardware benötigt wird. Das System ist weiterhin sehr robust gegenüber Umwelteinflüssen, da keine Modifikationen am Steuergerät vorgenommen werden müssen. Die Hardware, die hardwarenahe Software sowie das Betriebssystem sind identisch mit dem Zielsystem. Damit können die Ergebnisse problemlos im späteren Zielsystem weiterverwendet werden.

Als problematisch stellen sich beim „internen Bypass“ der häufig nur begrenzt vorhandene Speicherausbau (RAM, ROM) sowie die relativ geringe Rechenleistung von Seriensteuergeräten dar. Das hat zur Folge, dass eine detaillierte Systemanalyse erforderlich ist, um herauszufinden, ob das System die geforderten Randbedingungen erfüllen kann. In vielen Fällen ist das Entfernen von nicht dringend benötigten Softwareteilen der einzige Weg, genügend Ressourcen für die neuen Anforderungen zu bekommen. Zudem ist es bei diesem Ansatz nicht möglich, weitere I/O-Schnittstellen einzubinden. Der gesamte Entwicklungsprozess gestaltet sich deshalb deutlich aufwändiger als bei den zuvor beschriebenen Varianten, da dieser an den Entwicklungsprozess des Seriensteuergerätes angelehnt ist. Außerdem ist auch hier ein Freischnitt der Seriensoftware erforderlich. Für Mikrocontroller ohne Floating Point Unit (FPU) ist eine entsprechend aufwändige Integer-Codierung erforderlich.

Bewertung "Interner Bypass"

Vorteile:

- Kleine Bauform
- Keine zusätzliche Hardware erforderlich (falls RAM, ROM, I/O ausreichen)
- Robustes System, da keine zusätzliche externe Schnittstelle benötigt wird
- I/O identisch mit Zielsystem
- I/O-Softwaretreiber identisch mit Zielsystem
- Betriebssystem identisch mit Zielsystem
- Einfache Weiterverwendbarkeit der Ergebnisse

Nachteile:

- Wenig Speicher (RAM / FLASH)
- Geringe Rechenleistung (=> Detaillierte Analyse der Performance des Systems erforderlich)
- Keine Erweiterbarkeit der I/Os
- Lange Modifikationszeiten, da Codegenerierung und Flashen für das Steuergerät erforderlich ist
- Freischnitt in der Steuergerätesoftware erforderlich

4.3 Beurteilung der Entwicklungsmethoden

In Abbildung 4-11 sind die sehr wichtigen Beurteilungskriterien (Flexibilität, Performance, Identität mit dem Zielsystem) für Entwicklungssysteme dargestellt.

Man kann erkennen, dass ein Universal Experimentiersystem die höchste Flexibilität und Performance bietet, dafür aber im Bereich der Zielsystemidentität und damit bei der Weiterverwendbarkeit schlecht abschneidet. Diese Systeme sind demzufolge für Grundsatzuntersuchungen ohne direkten Serienbezug, bei denen noch sehr viele unterschiedliche Ansätze betrachtet werden müssen, zu empfehlen.

Der „interne Bypass“ bietet den Lösungsansatz, der dem Zielsystem am nächsten kommt. Dafür sind hier jedoch die Performance und Flexibilität sehr eingeschränkt. Durch die aufwändigere Codegenerierung sind die Rekursionszeiten bei der Entwicklung deutlich größer als beim externen Bypass. Dieser Ansatz eignet sich am besten, wenn die Hardware bereits komplett und große Teile der Software ebenfalls fixiert sind.

Beim „externen Bypass“ ist die Identität durch die Bauform des Gesamtsystems etwas geringer. Dafür kann aber die Performance durch den Einsatz eines Simulationsrechners gesteigert werden. Zudem kann die Flexibilität durch Verwendung von Standard-I/O-Interfaces im Bypass-System ebenfalls erhöht werden. Dieser Lösungsweg wird häufig gewählt, wenn zu bestehenden Steuergeräteständen neue Konzepte zu einzelnen Funktionen entwickelt werden sollen.

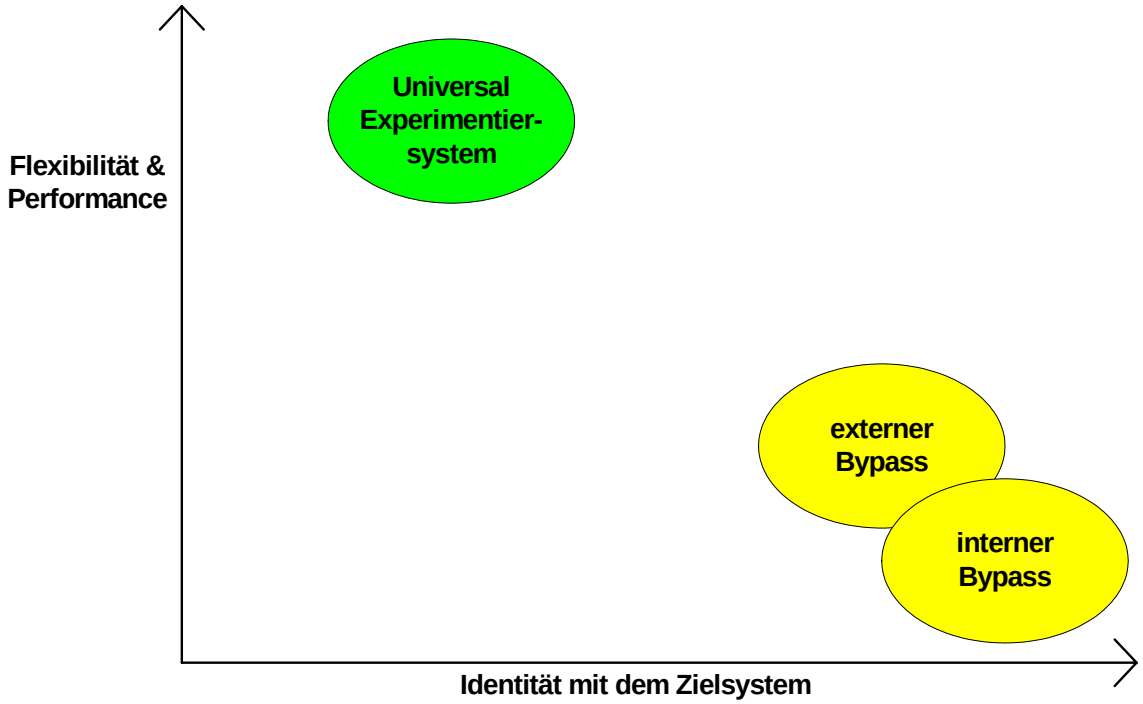


Abbildung 4-11: Vergleich verschiedener Entwicklungsmethoden

Kapitel 5

KONZEPTION UND STRUKTUR EINES NEUEN ANSATZES

5 Konzeption und Struktur eines neuen Ansatzes

In diesem Abschnitt wird ein Konzept zur effizienten Umsetzung der in Abschnitt 2 definierten Ziele vorgestellt. Aus den bereits aufgeführten Gründen konzentriert sich dieser Ansatz vorwiegend auf die Informations- und Elektrotechnik. Diese Fokussierung ist in Abbildung 5-1 am mehrfach durchlaufenen V-Modell (vergl. Abschnitt 3.1.2 und 3.1.4) dargestellt. Die Bereiche, auf denen in dieser Arbeit das Hauptaugenmerk liegt, sind mit einem kräftigen Blau dargestellt, wohingegen die weniger betrachteten Bereiche in hellerem Blau bis hin zu Weiß abgebildet sind.

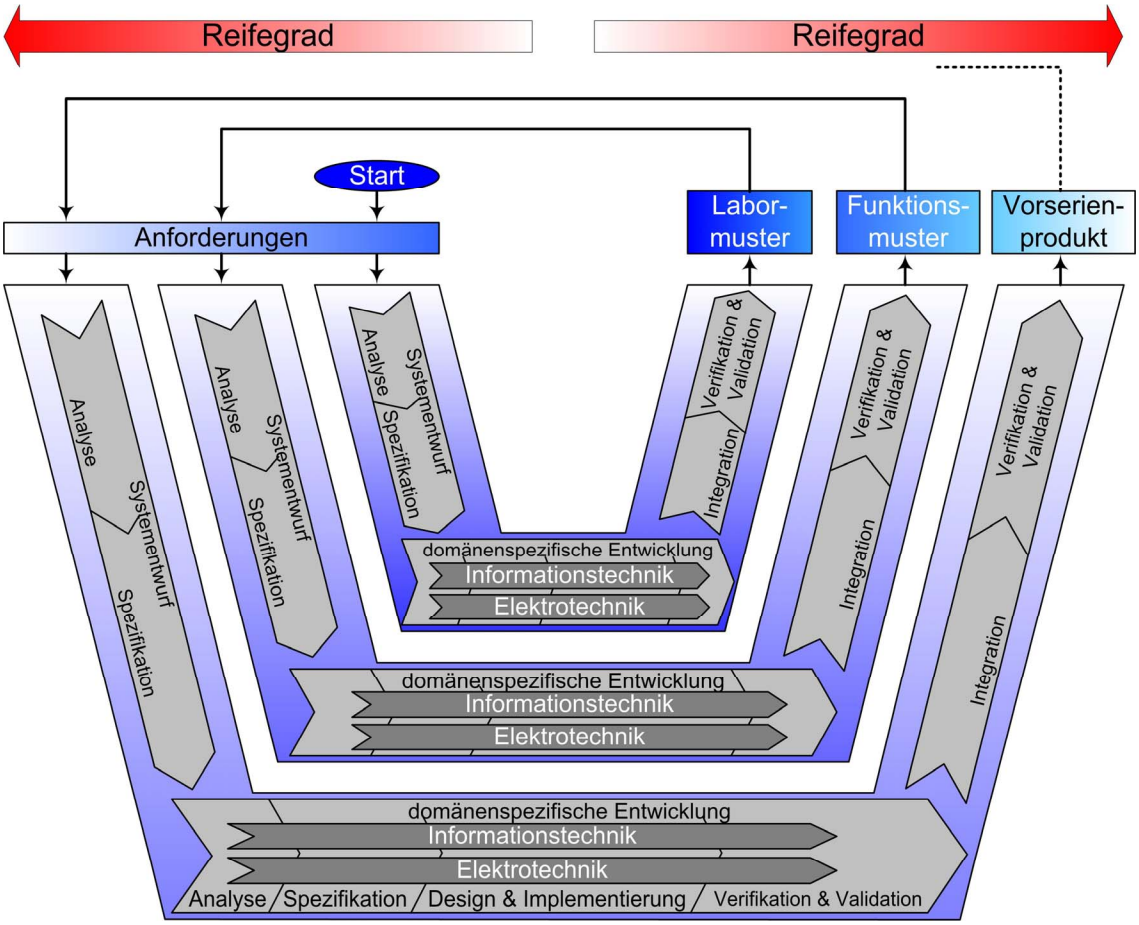


Abbildung 5-1: Einordnung im V-Modell

Mit diesem Ansatz soll, wie Abbildung 5-1 entnommen werden kann, nicht auf sehr frühe Schritte der Entwicklung eingegangen werden. Hierfür können beispielsweise die ersten Phasen des modellbasierten Systementwurfs (Zielformulierung, Modellbildung, Modellanalyse) verwendet werden. Dieser Ansatz ist so angelegt, dass er sich einfach in die Systemsynthese und Systemanalyse des modellbasierten Systementwurfs (vergl. Abschnitt 3.2) einfügt. So kann mit diesem Ansatz unter anderem nach erfolgreichem Funktionsnachweis z.B. mit Software-in-the-Loop (SIL) die Korrektheit der Funktion am realen System nachgewiesen werden sowie durch Kopplung des realen Systems mit dem virtuellen Streckenmodell die Fehlerfreiheit der Software und der Schnittstellen überprüft werden (Hardware-in-the-Loop (HIL)). Des Weiteren kann bei einer ersten Integration die Funktion am realen System in Betrieb genommen, überprüft und in einer Feinabstimmung optimal an die realen Verhältnisse angepasst werden. Neben dem oben schon erwähnten V-Modell können auch weitere Bestandteile der in [VDI03] vorgeschlagenen „Entwicklungsmethodik Mechatronik“ (vergl. 3.1.4) wie der Problemlösezyklus als Mikrozyklus und Prozessbausteine für wiederkehrende Arbeitsschritte verwendet werden.

Um Systeme mit ständig zunehmender Komplexität in immer kürzeren Entwicklungszeiten bereitstellen zu können, ist es unter anderem von großer Bedeutung, Erfahrungen aus bereits laufenden Serienprojekten zu berücksichtigen. Diese Erfahrungen können in Form von wieder verwendeten Komponenten sowohl aus den Bereichen der Simulation (Modelle), der Hardware, als auch der Software kommen.

Durch die Wiederverwendung der Erfahrungen aus früheren Serienprojekten in einem neuen Entwicklungssystem kann aber nicht nur dieses Entwicklungssystem schneller dargestellt werden, sondern es ist auch deutlich einfacher möglich, die mit diesem System dargestellten Lösungen erneut in eine Serienlösung zu überführen. Die Wiederverwendung von Erfahrungen und Ergebnissen aus Serienentwicklungen war bislang den Systemlieferanten vorbehalten. Mit diesem Ansatz werden diese Erfahrungen nun auch den OEM sowie Spezialfirmen, welche spezielles Know-How bezüglich des Einsatzes bestimmter Technologien haben, zugänglich. Dadurch werden beispielsweise sehr effiziente Möglichkeiten für das Simultaneous Engineering (vergl. 3.4) zwischen OEM, Systemlieferant und Spezialfirma eröffnet.

Weitere sehr wichtige Aspekte sind der konsequente Einsatz von Standards der Softwareentwicklung der Fahrzeugindustrie wie beispielsweise CARTRONIC (vergl. Abschnitt 3.5.1), OSEK/VDX (vergl. 3.5.2) und ASAM (vergl. 3.5.3) sowie generell die Steigerung des Automatisierungsgrades von Entwicklungsprozessen.

Im Folgenden wird die Konzeption, aufgeteilt in die Teile Informations- und Elektrotechnik, genauer dargestellt.

5.1 Konzeption des Informationstechnikanteils (Software)

5.1.1 Softwarestruktur des Entwicklungssystems

Ausgangsbasis für die Softwarestruktur des im Rahmen dieser Arbeit konzipierten und realisierten Entwicklungssteuergerätes ist die Softwarestruktur eines Seriensteuergerätes, wie sie in Abbildung 5-2 zu sehen ist. Wesentliche Bestandteile der auf dem Mikrocontroller ausgeführten Software sind neben der Anwendersoftware das Echtzeitbetriebssystem, die Services sowie die Hardwaretreiber. Die Services sind unterstützende und sehr gut optimierte Routinen für arithmetische Berechnungen, zum Filtern von Signalen sowie zur Integration und Interpolation. Die Hardwaretreiber kapseln die Zugriffe der Anwendersoftware auf die Hardware und stellen Schnittstellen z.B. für das Auslesen von A/D-Wandlern, das Lesen und Schreiben digitaler und pulsweitenmodulierter Ein-/ Ausgänge und die Kommunikation z.B. über den CAN Bus zur Verfügung. Die Kombination aus Betriebsystem, Services und Hardwaretreiber werden bei eingebetteten Systemen häufig auch als „**H**ardware **A**bstraction **L**ayer“ (HAL) bezeichnet. Die Schnittstellen zwischen den einzelnen Bestandteilen sind in den folgenden Abbildungen als rote Dreiecke dargestellt.

Quelle: [GHB98]

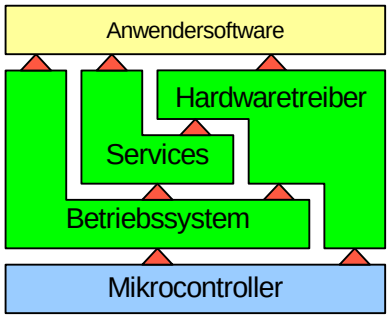


Abbildung 5-2: Softwarestruktur eines Seriensteuergerätes

5.1.2 Auslagerung der Anwendersoftware des Mikrocontrollers

Die Leistungsfähigkeit des Entwicklungssystems wird durch die Einbindung eines Simulationsprozessors zu höheren Leistungen skalierbar. Durch die Leistungsfähigkeit des Simulationsprozessors kann die Anwendung beispielsweise komplett und schnell in Fließkomma-Arithmetik entwickelt und getestet werden. Dazu wird die für den Einsatz in einem Ein-Prozessor-System entworfene Softwarestruktur so erweitert, dass neben dem Mikrocontroller auch der Simulationsprozessor einfach eingebunden werden kann. Die daraus abgeleitete Softwarearchitektur des Zwei-Prozessor-Systems ist in Abbildung 5-3 dargestellt. Die Anwendersoftware ist in diesem Fall komplett auf den leistungsfähigen Simulationsprozessor ausgelagert.

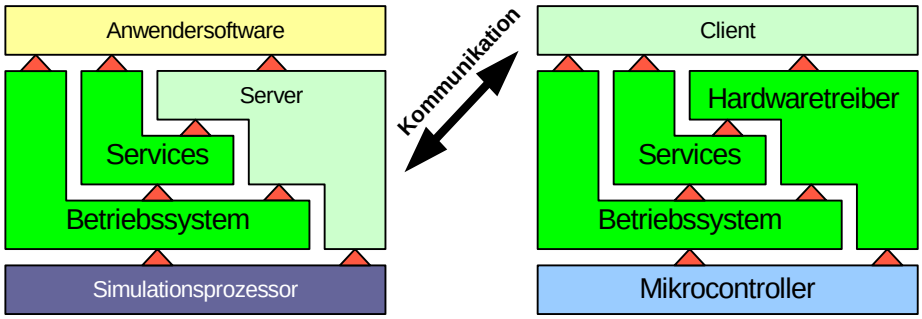


Abbildung 5-3: Softwarestruktur des Entwicklungssystems I

Die Auslagerung der Anwendersoftware auf den Simulationsprozessor wird durch die Einführung einer Kommunikationsschnittstelle ermöglicht. Über diese Kommunikationsschnittstelle stellt der Client dem Server die benötigten Eingangsgrößen zur Verfügung bzw. holt die Ausgangsgrößen von diesem ab. Auf die Kommunikationsschnittstelle wird in Abschnitt 5.1.5 näher eingegangen.

Betrachtet man das Schichtenmodell der Softwarearchitektur des Simulationsprozessors, so kann man erkennen, dass die Anwendersoftware lediglich auf das Betriebssystem, die Services sowie den Server zugreift. Da das Betriebssystem und die Services von Simulationsprozessor und Mikrocontroller so weit als möglich identisch sind (siehe dazu auch Abschnitt 5.1.3), muss nur noch vom Server sichergestellt werden, dass sich die Schnittstelle zur Anwendersoftware identisch zu der Schnittstelle der hardwarenahen Software verhält. Das wird durch die Anlehnung der Funktions- und Namensstruktur an CARTRONIC erreicht. Auf diese Weise ist es sehr einfach möglich, die Anwendersoftware auf dem Simulationsrechner zu entwickeln und anschließend schrittweise oder auch in einem Schritt auf den Mikrocontroller zu verschieben (Abbildung 5-4). In den meisten Fällen wird beim Übergang auf den Mikrocontroller ebenfalls auf Integer-Arithmetik umgestellt.

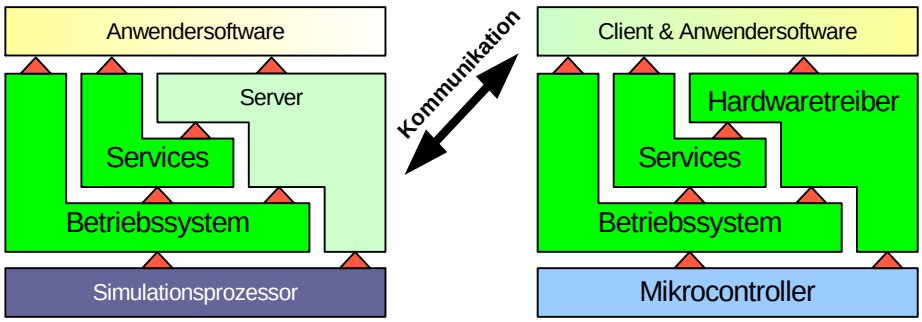


Abbildung 5-4: Softwarestruktur des Entwicklungssystems II

Wenn die komplette Anwendersoftware auf den Mikrocontroller verschoben wurde, kann das System auch ohne Simulationsprozessor betrieben werden. Dadurch ist es möglich, Bauraum, Verlustleistung und Kosten einzusparen. Die Softwarestruktur entspricht dann wieder Abbildung 5-2.

5.1.3 Verwendung von Seriensoftware im Entwicklungssystem

Damit das Entwicklungssystem möglichst ein mit dem Zielsystem vergleichbares Verhalten aufweist und somit die Ergebnisse leichter auf das Zielsystem übertragen werden können, werden möglichst viele Teile aus der Serienentwicklung verwendet. Wie im Schichtenmodell der Softwarearchitektur des Mikrocontrollers in Abbildung 5-4 zu erkennen ist, laufen dort das Betriebssystem, die Services, die Hardwaretreiber sowie der Client und evtl. ein Teil oder die gesamte Anwendersoftware. Abgesehen vom Client und der Anwendersoftware können diese Softwarekomponenten direkt aus Serienprojekten wieder verwendet werden. Als Betriebssystem wird ein OSEK konformes Echtzeitbetriebssystem eingesetzt, bei OSEK handelt es sich um einen im Fahrzeugbereich anerkannten und de facto Standard (vergl. Abschnitt 1.5.2).

Da bei Steuergeräten für den Antriebsstrang die vergleichsweise größten Anforderungen an Rechenleistung und I/O Umfang bestehen, werden die Hardwaretreiber und Services aus Serienprojekten dieses Bereichs verwendet.

Auf dem Simulationsprozessor kommen entsprechend portierte Varianten des auf dem Mikrocontroller verwendeten Echtzeitbetriebsystems sowie der Services zum Einsatz.

5.1.4 Echtzeitfähigkeit des Entwicklungssystems

Für die Steuerung und Regelung von Systemen in Kraftfahrzeugen werden sowohl mehrere durch zeitliche Bedingungen getriggerte Rechenraster, als auch durch die Position der Kurbelwelle (KW) getriggerte Rechenraster benötigt. Übliche Rechenraster sind:

- 1 ms, 5 ms, 10 ms, 20 ms, 100 ms => zeitsynchron
- 180 ° KW, 360 ° KW, 720 ° KW => winkelsynchron

Der Mikrocontroller ist der Master sowohl für die zeitsynchronen, als auch für die winkelsynchronen Rechenraster, da dieser direkt mit der Hardware, welche die Kurbelwellenwinkel auswertet, in Verbindung steht. Die Zeitraster werden vom Betriebssystem über einen Timer des Mikrocontrollers bereitgestellt. Die winkelsynchronen Raster werden durch externe Interrupts realisiert. Diese externen Interrupts können durch eine rekonfigurierbare Logik flexibel konfigurierbar zur Verfügung gestellt werden. Der Mikrocontroller kann den Simulationsprozessor durch einen Interrupt triggern und somit die entsprechenden Rechenraster auch auf diesem zur Verfügung stellen. Die Hard- und Software-Implementierung des Entwicklungssystems muss sicherstellen, dass Zeitverschiebungen oder Zeitschwankungen dabei minimiert werden. In Abbildung 5-5 ist das exemplarisch für ein Rechenraster dargestellt.

5.1.5 Kommunikationsmechanismus und Sicherheitskonzept

Die Kommunikation zwischen dem Mikrocontroller und dem Simulationsprozessor läuft nach einem im Rahmen dieser Arbeit konzipierten und an das DISTAB-Verfahren (vergl. Abschnitt 4.2.1) angelehnten Mechanismus ab. Der Ablauf für ein Raster ist exemplarisch in Abbildung 5-5 dargestellt.

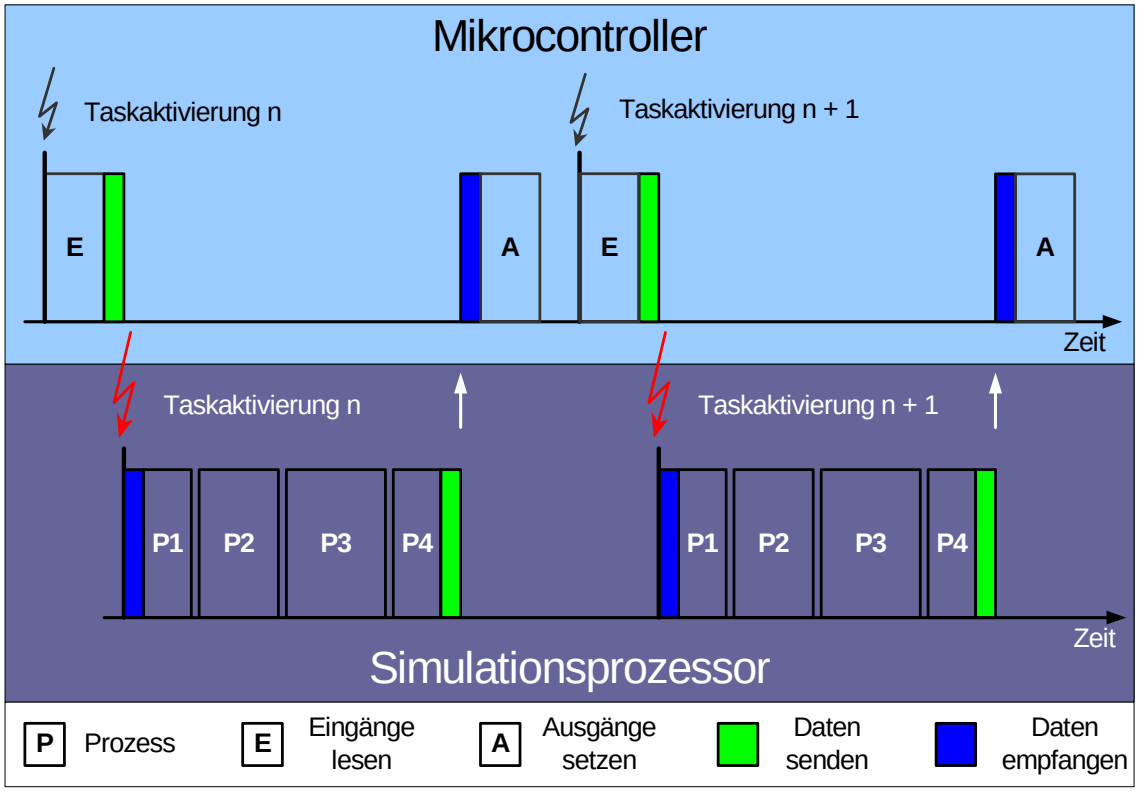


Abbildung 5-5: Kommunikationsablauf

Der Mikrocontroller liest die Eingänge von der Hardware ein und kommuniziert die so ermittelten Eingangsdaten an den Simulationsprozessor. Der Simulationsprozessor berechnet daraus die benötigten Werte und kommuniziert diese wiederum zurück zum Mikrocontroller. Der Mikrocontroller nimmt die Werte entgegen, prüft diese gegebenenfalls auf Plausibilität und steuert anschließend die Hardware an. Bei dem Plausibilitätstest wird geprüft, ob die Kommunikation zwischen Simulationsprozessor und Mikrocontroller aktiv ist, das entsprechende Signal vom Simulationsprozessor bereitgestellt wurde und ob dieses Signal in einen gültigen Wertebereich liegt. Ist nur eine Bedingung nicht erfüllt, kann abhängig vom erforderlichen Sicherheitskonzept ein Ersatzwert verwendet oder das System deaktiviert werden. Ein Ersatzwert kann entweder eine Konstante sein oder aber auch vom Mikrocontroller selbst berechnet werden.

5.1.6 Softwareentwicklung und Codegenerierung

Sowohl die Anwendersoftware für den Mikrocontroller, als auch die Anwendersoftware für den Simulationsprozessor, werden mit demselben grafischen Entwicklungswerkzeug erstellt. Durch die Verwendung einer einheitlichen Entwicklungsumgebung wird das Verschieben einzelner Funktionen zwischen Mikrocontroller und Simulationsprozessor erheblich erleichtert. Die Verwendung einer grafischen Entwicklungsumgebung ermöglicht es, für jede Aufgabe die am besten passende Beschreibungsform zu verwenden:

- Blockdiagramm,
- Zustandsautomat,
- Textuelle Programmiersprache (z.B. „C“ [KP93]).

Die Erstellung der Anwendersoftware ist nicht Schwerpunkt dieser Arbeit. Die Anwendersoftware soll von den Anwendern dieses Ansatzes erstellt werden. Es empfiehlt sich allerdings, diese mit Hilfe des modellbasierten Systementwurfs zu entwickeln (siehe Abschnitt 3.2). Dieser Ansatz unterstützt die Adaption, erleichtert die Umsetzung

(Abbildung 3-7) und konzentriert sich somit auf die Phasen Systemsynthese und Systemanalyse (Abbildung 3-5).

Bei der Softwareentwicklung für den Mikrocontroller werden das Betriebsystem, die Hardwaretreiber und die Services in Form von vorkompilierten Archiven dazu gebunden. Der Client und evtl. vorhandene Teile der Anwendersoftware werden in den von der Entwicklungsumgebung bereitgestellten Beschreibungsformen spezifiziert. Aus allen Einzelbestandteilen wird mittels automatischer Codegenerierung und anschließendem Kompilier- und Linkvorgang ein ausführbares Programmfile (*.s19) und eine Beschreibungsdatei nach dem ASAM-MCD-2 Standard (*.a2l) generiert. Durch die Definition und Verwendung angepasster Modellierungsrichtlinien und der Nachbearbeitung der ASAM-MCD-2 Datei, stehen unter anderem genaue Informationen zu den Größen bereit, welche vom Mikrocontroller zum Simulationsprozessor und umgekehrt übertragen werden können. Mit diesen Informationen kann die Verbindung zwischen dem Mikrocontroller und dem Simulationsprozessor hergestellt werden. Das ausführbare Programm (*.s19) kann z.B. mit Hilfe eines Debuggers auf dem Mikrocontroller ausgeführt werden.

Bei der Softwareentwicklung für den Simulationsprozessor werden sowohl der Server, als auch die Anwendersoftware in den von der Entwicklungsumgebung bereitgestellten Beschreibungsformen spezifiziert. Mit Hilfe der nachbearbeiteten ASAM-MCD-2 Datei des Mikrocontrollers kann eine Verbindung von Größen des Mikrocontrollers, mit denen des Simulationsprozessors, hergestellt werden. Auch hier wird aus den Einzelbestandteilen mittels automatischer Codegenerierung und anschließendem Kompilier- und Linkvorgang ein ausführbares Programm erstellt. Dieses Programm kann direkt mit Hilfe der Entwicklungsumgebung auf dem Simulationsprozessor ausgeführt werden.

In Abbildung 5-6 ist die Entwicklungsumgebung mit den entsprechenden Ein- und Ausgabedateien zu sehen.

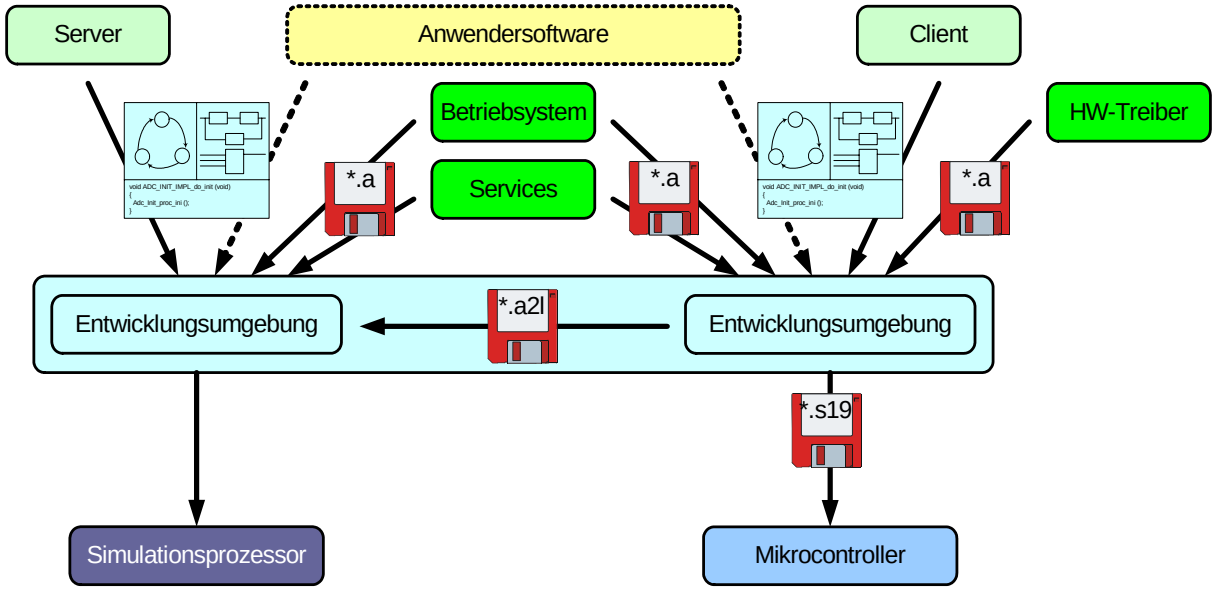


Abbildung 5-6: Entwicklungsumgebung

5.2 Konzeption des Elektrotechnikteils (Hardware)

5.2.1 Partionierung in unabhängige Teilkomponenten

Ausgangsbasis für die Hardwarestruktur des Entwicklungssteuergerätes ist ebenfalls die Struktur eines Seriensteuergerätes des Antriebstrangs, wie sie in Abbildung 5-7 zu sehen

ist. Wesentliche Bestandteile sind dabei der Mikrocontroller, der bereits einen nicht unerheblichen Anteil der Peripheriefunktionen integriert hat. Externe Speicherbausteine, zum Ablegen flüchtiger und nichtflüchtiger Informationen, sowie einige anwendungsspezifische ASIC's und auf die Anwendung optimierte diskret aufgebaute Signalkonditionierungen. Alle Signale von und zur Peripherie werden über das „**V**ehicle **C**ontrol **I**nterface" (VCI) geführt. Das VCI von Seriensteuergeräten ist in der Regel auf einige wenige Steckzyklen begrenzt, da der Steckverbinder nur zur Fehlersuche entfernt werden muss.

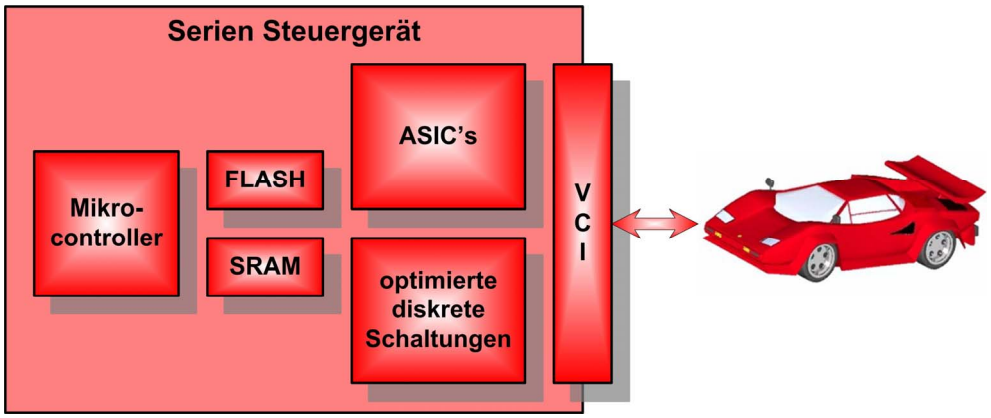


Abbildung 5-7: Hardwarestruktur eines Seriensteuergerätes

Diese extrem auf Kosten optimierte Einheit wird im ersten Schritt in zwei voneinander nahezu unabhängige Komponenten aufgeteilt. Diese Aufteilung wird so vorgenommen, dass zum einen der Mikrocontroller samt Speicher eine Komponente bilden (Mikrocontroller Board) und zum anderen die Signalkonditionierung und Leistungstreiber (Adaptation Board), welche bestimmend für die konkrete Anwendung sind, ebenfalls eine Einheit bilden. Durch diese Aufteilung ist es möglich, die beiden Teile unabhängig und in unterschiedlichen Zeitintervallen weiterzuentwickeln. Somit kann das Mikrocontroller Board einfach für unterschiedliche Anwendungen eingesetzt werden. Oder es kann ein Redesign vorgenommen werden, um z.B. mehr Speicher oder einen Controller mit schnellerem Systemtakt zur Verfügung zu stellen, ohne das Gesamtsystem verwerfen zu müssen. Dieses optimierte Mikrocontroller Board kann dann zusammen mit allen im Laufe der Zeit entstehenden Adaptation Boards eingesetzt werden, ohne dass diese selbst einem Redesign unterzogen werden müssen. Damit das Zusammenwirken der Mikrocontroller- und Adaptation Boards funktionieren kann, ist es erforderlich, eine geeignete Schnittstelle zu spezifizieren. Diese sowie weitere Schnittstellen werden in den nächsten Abschnitten beschrieben.

5.2.2 Mikrocontroller Board

Das zentrale Element dieses Boards ist, wie der Name schon verrät, der Mikrocontroller. Hier kommt entweder der Mikrocontroller des Zielsystems oder ein diesem sehr ähnlicher Mikrocontroller zum Einsatz. Das ist erforderlich, da moderne Mikrocontroller bereits einen nicht unerheblichen Anteil der Peripheriefunktionen integriert haben und deren Nachbildung extrem aufwändig wäre. Dieser Mikrocontroller wird im Gegensatz zu einem Seriensteuergerät in großem Maße mit schnellem SRAM- und FLASH- Speicher ausgestattet.

Um sehr schnell Daten z.B. mit einem Simulationsprozessor austauschen zu können, wird eine entsprechende Schnittstelle „**S**imulation **P**rocessor **I**nterface" (SPI) vorgesehen. Diese Schnittstelle sollte auf einem standardisierten Bus basieren, damit auf eine große Menge bestehender Simulationsprozessoren zurückgegriffen werden kann.

Zudem wird noch eine Schnittstelle zum Anschluss eines Debuggers „**M**ikrocontroller **D**ebug **I**nterface“ (MDI) vorgesehen. Über diese Schnittstelle ist es möglich, das Programm in das SRAM oder FLASH des Mikrocontrollers zu laden und beispielsweise die Ausführung im Einzelschrittbetrieb oder über Breakpoints zu steuern.

Sämtliche Peripherie-Signale des Mikrocontrollers sowie der komplette Adress-, Daten- und Control-Bus werden auf einem nach Funktionsgruppen geordneten Stecker, dem „**A**daptation **B**oard **I**nterface“ (ABI), zur Verfügung gestellt. An dieses Interface können dann unterschiedlichste aufgabenspezifische Adaptation Boards angeschlossen werden.

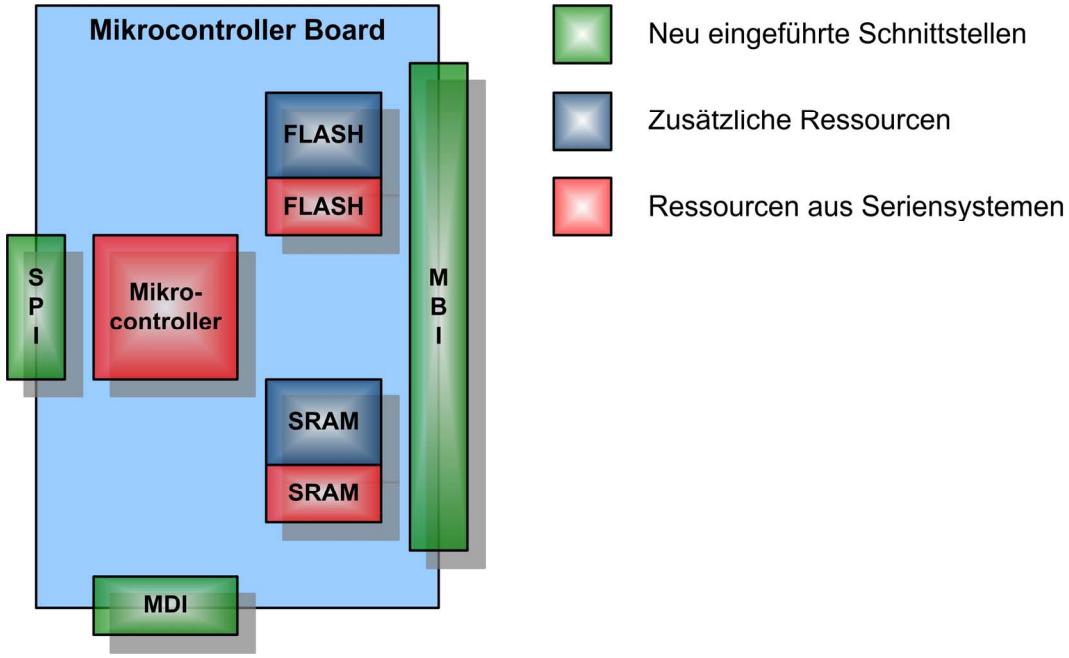


Abbildung 5-8: Mikrocontroller Board

Das Mikrocontroller Board ist durch seinen leistungsfähigen Mikrocontroller, die große Menge an Speicher und die universellen Schnittstellen für eine Vielzahl unterschiedlichster Anwendungen einsetzbar.

5.2.3 Adaptation Board

Um im Bereich der Signalgenerierung und Signalauswertung die Flexibilität zu steigern, ohne die Identität mit dem angestrebten Zielsystem zu verlieren, werden auf den Adaptationboards soweit möglich ASIC’s und andere Halbleiter aus dem Zielsystem verwendet, diese aber nicht direkt mit dem Mikrocontroller gekoppelt. Anstelle der direkten Kopplung wird, wie in Abbildung 5-9 zu sehen ist, eine leistungsfähige **R**ekonfigurierbare **L**ogik (RL) dazwischengeschaltet. Somit ist es für einen schnellen Start der Entwicklung möglich, eine Konfiguration vergleichbar mit einem bestehenden Seriensystem herzustellen. Weitergehende Anforderungen, wie beispielsweise neue Protokolle auf den Kommunikationskanälen (TT-CAN), spezielle Signalauswertungen (für Drehzahl-, Positionserfassung, neue Sensoren, etc.) oder Signalgenerierungen (für Einspritz-steuerung, Ventilansteuerung, etc.), können sehr einfach und flexibel in der RL realisiert werden. Durch den Einsatz rekonfigurierbarer Logik wird es ermöglicht, sehr einfach und schnell Änderungen und Erweiterungen in der normalerweise statischen Hardware vorzunehmen. Zusätzlich kann durch Einbetten des Mikrocontrollers in die rekonfigurierbare Logik die normalerweise sehr früh zu treffende Entscheidung, ob eine Funktion im Mikrocontroller, in der rekonfigurierbaren Logik oder aufgeteilt auf beide Einheiten realisiert werden soll, noch in sehr fortgeschrittenen Entwicklungsabschnitten beeinflusst werden. Die in der rekonfigurierbaren Logik umgesetzten Funktionen können

z.B. als ausführbare Spezifikation für ASIC Entwicklungen verwendet werden. Es können somit sehr einfach unterschiedliche Integrationsstufen (siehe Abschnitt 1.1) realisiert und real erprobt werden. Begünstigt durch die in den letzten Jahren anhaltende Preisreduktion bei rekonfigurierbarer Logik wird es zukünftig evtl. möglich sein, diese direkt im Seriensystem einzusetzen. Dadurch können die heute üblichen ASIC's, welche sowohl digitale, als auch analoge Komponenten beinhalten, als reine analoge ASIC's ausgeführt werden und damit eine für analoge Belange optimale Technologie eingesetzt werden.

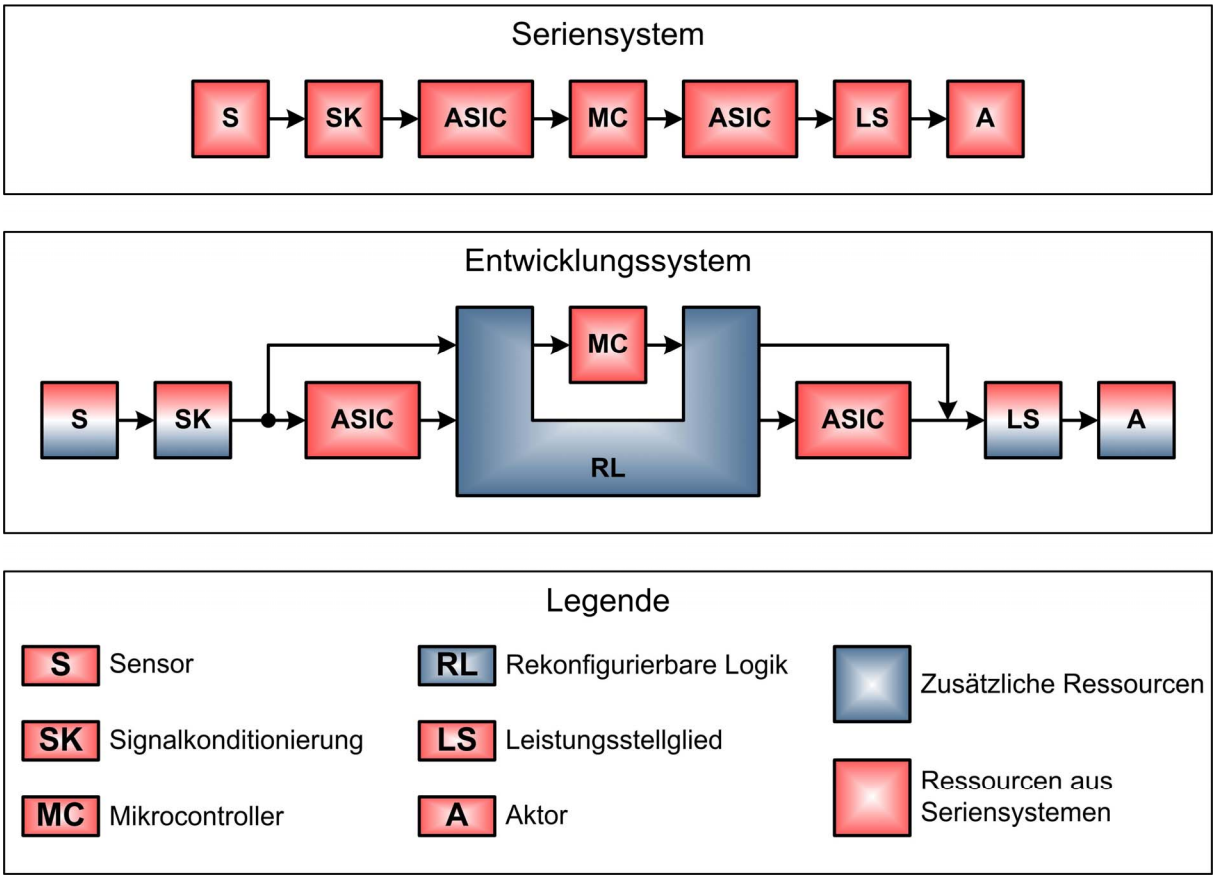


Abbildung 5-9: Integration der regkonfigurierbaren Logik

Auf den aufgabenspezifischen Adaptation Boards wird noch eine weitere Schnittstelle, das „**E**xpansion **B**oard **I**nterface“ (EBI), implementiert. Reichen die Ressourcen auf dem Adaptation Board selbst nicht aus, so ist eine Erweiterung über das EBI, ohne Modifikation des Adaptation Board, jederzeit möglich. Auf dem Expansion Board können beispielsweise weitere schnelle analoge Eingänge über DSP's vorverarbeitet und dem Mikrocontroller zur Verfügung gestellt werden. Daneben können über die Integration einer leistungsfähigen rekonfigurierbaren Logik z.B. in Form eines großen FPGA weitere IP Cores inkl. Peripherie genutzt und z.B. für Untersuchungen zum HW/SW Codesign verwendet werden. Die Signale des Expansion Boards können ebenfalls über das „**V**ehicle **C**ontrol **I**nterface“ (VCI) geführt werden.

Um die teilweise sehr komplexen Inhalte der RL schnell entwickeln und debuggen zu können, wird auch hierfür eine Schnittstelle zur Verfügung gestellt, das „**R**ekonfigurable Logic **D**ebug **I**nterface“ (RDI).

Die Abbildung 5-10 zeigt den schematischen Aufbau eines Adaptationboards.

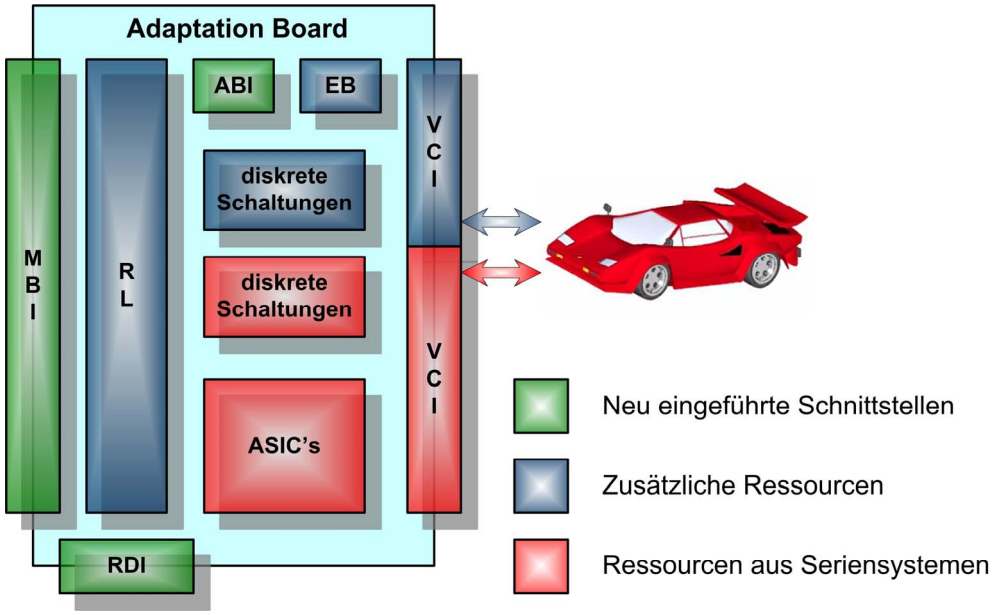


Abbildung 5-10: Adaptation Board

In Abbildung 5-11 sind die Komponenten zu einem Gesamtsystem zusammengefügt. Der Anwender hat dabei sehr gute Eingriffsmöglichkeiten über die beiden Debug Schnittstellen (MDI, RDI) sowie über das „**H**ost **U**ser **I**nterface“ (HUI) des Simulation Prozessor Boards. Für Anwendungen, bei denen der Umfang eines einzigen VCI nicht ausreichend ist, können mehrere Mikrocontroller Boards mit den jeweiligen Adaptation Boards von einem einzigen Simulationsprozessor kontrolliert werden. Auf das Simulations Prozessor Board wird nicht näher eingegangen, da dieses nicht im Rahmen dieser Arbeit entworfen und realisiert wurde. Als Simulations Prozessor Board wird nachfolgend ein kommerziell verfügbares Produkt wie es beispielsweise von der ETAS GmbH angeboten wird eingesetzt.

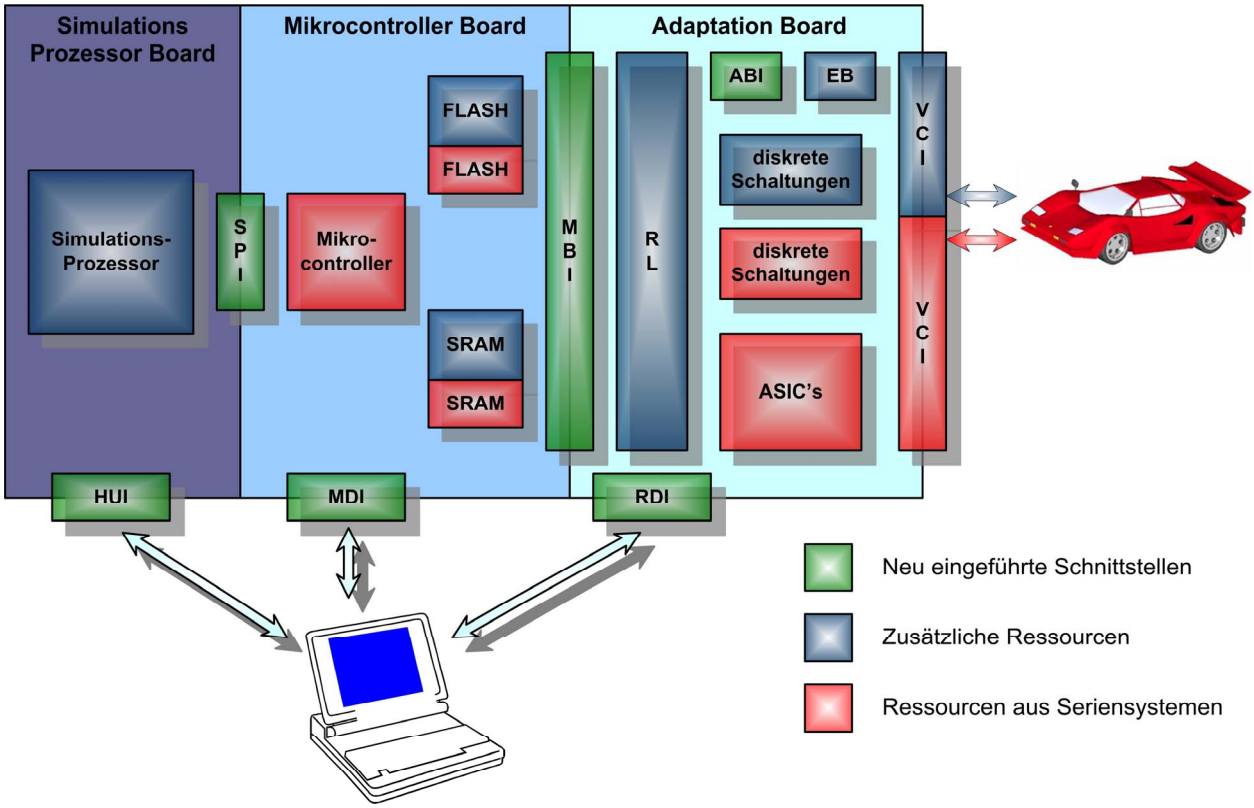


Abbildung 5-11: Komponenten des Gesamtsystems

5.3 Einordnung im Bezug zu alternativen Lösungsansätzen

In diesem Abschnitt werden bestehende Lösungsansätze dargestellt, kurz analysiert sowie deren Vor- und Nachteile zusammengefasst. Betrachtet werden die „MicroAutoBox“ von dSPACE, der IAV „Engine Controller“ sowie der RICARDO „Network Vehicle Controller“.

5.3.1 Die dSPACE MicroAutoBox

Die MicroAutoBox ist ein kompaktes zwei Prozessor Prototypingsystem. Die Softwareentwicklung kann mit dem grafischen Entwicklungswerkzeug MATLAB / Simulink durchgeführt werden. Als Kommunikationsschnittstellen bietet das System CAN, K-Line, RS232, LIN oder optional FlexRay. Die analogen Schnittstellen haben einen Spannungsbereich von 0-5V. Als digital I/O werden nur TTL Signale unterstützt was einen direkten Einsatz am realen System nahezu unmöglich macht. Es ist geplant über zusätzliche Boxen (RapidPro) Fahrzeugtaugliche I/O zur Verfügung zu stellen. Dadurch wird allerdings die Bauform deutlich vergrößert und die Ansteuerung von zeitkritischen Komponenten wie z.B. von Injektoren wird dennoch nicht möglich sein, da die dafür benötigten intelligenten Peripheriebausteine oder entsprechende Nachbildungen der Funktionalitäten mit Hilfe von rekonfigurierbarer Logik nicht vorhanden sind. Eine Erweiterung der Rechenleistung durch einen zusätzlichen Simulationsprozessor ist nicht vorgesehen. Eine flexible Signalerfassung, Signalauswertung oder Signalausgabe z.B. mit Hilfe einer rekonfigurierbaren Logik wird nicht unterstützt. Ebenfalls ist die Anbindung zusätzlicher schneller Schnittstellen nicht möglich. Die Verwendung von serienerprobten Softwarekomponenten oder ASIC’s wird nicht angeboten. [Dsp04]

Quelle: [Dsp04]

Abbildung 5-12: dSPACE MicroAutoBox

5.3.2 Der IAV Engine Controller

Der IAV Engine Controller ist ein kompaktes zwei Prozessor Prototyping Entwicklungssteuergerät. Als Prozessoren kommen der Motorola Mikrocontroller MPC565 sowie der Infineon TriCore Mikrocontroller zum Einsatz. Beide Prozessoren sind auf dem aktuellen Stand der Technik. Die Softwareentwicklung kann z.B. über ASCET sowie MATLAB / Simulink / Targetlink durchgeführt werden. Als Betriebsystem wird das im Automobilbereich erprobte Echtzeitbetriebsystem ERCOS^{EK} verwendet. Die Ansteuerung von Einspritzventilen ist möglich. Eine flexible Signalerfassung, Signalauswertung oder Signalausgabe z.B. mit Hilfe einer rekonfigurierbaren Logik wird nicht unterstützt.

Ebenfalls ist die Anbindung zusätzlicher schneller Schnittstellen nicht möglich. Die Verwendung von serienerprobten Softwarekomponenten oder ASIC’s wird derzeit nicht angeboten. [RG+03]

Quelle: [IAV]

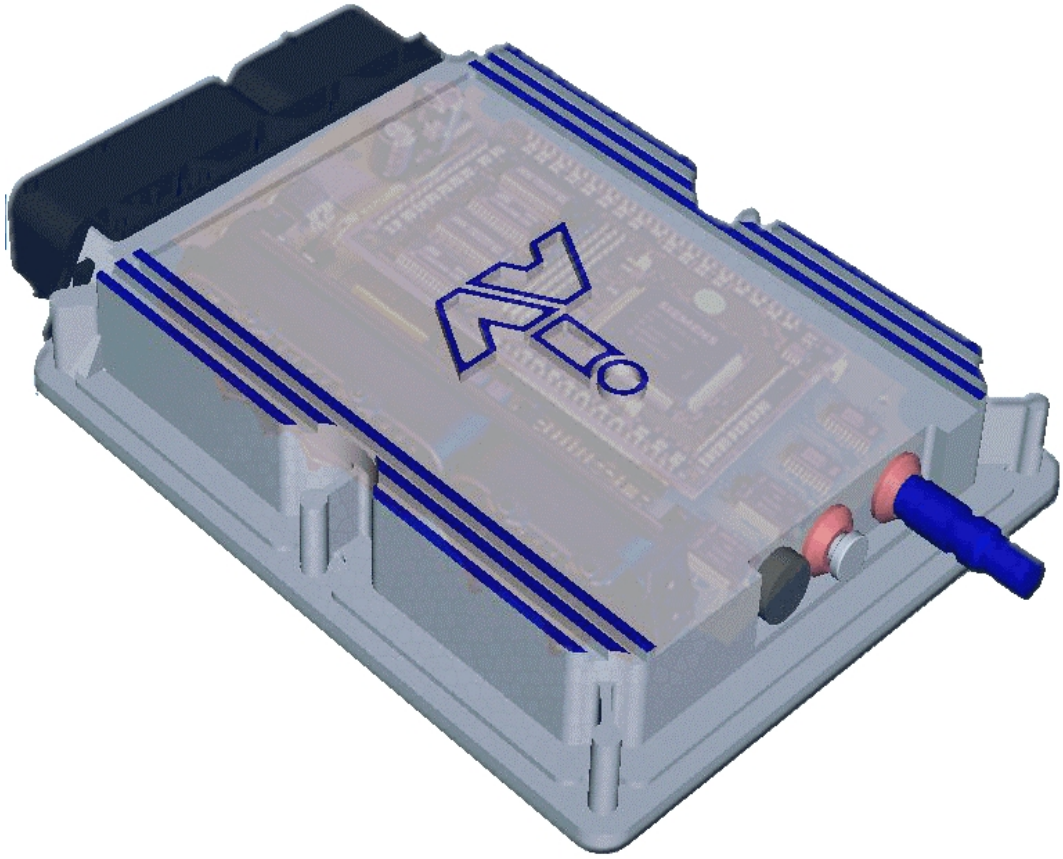


Abbildung 5-13: IAV Engine Controller

5.3.3 Der RICARDO Network Vehicle Controller

Der RICARDO Network Vehicle Controller ist ein noch kompaktes ein / zwei Prozessor Prototyping Entwicklungssteuergerät. Als Prozessoren kommen der Motorola Mikrocontroller MPC565 sowie optional der Motorola Application Prozessor MPC8260 zum Einsatz. Die Kommunikation zwischen beiden Prozessoren erfolgt über eine serielle SPI Schnittstelle, welche eine nur eingeschränkte Datenrate zur Verfügung stellen kann. Das System verfügt über eine ausreichende Anzahl an Fahrzeugtauglichen analogen und digitalen Ein- und Ausgängen sowie Kommunikationsschnittstellen. Allerdings sind keine Ausgänge zur direkten Ansteuerung von Einspritzventilen vorhanden. Die Softwareentwicklung für beide Prozessoren kann mit einer von Ricardo modifizierten Version von ASCET durchgeführt werden. Als Betriebsystem wird ein echtzeitfähiges Linux Derivat verwendet welches nicht in Serienanwendungen erprobt wurde. Eine flexible Signalerfassung, Signalauswertung oder Signalausgabe z.B. mit Hilfe einer rekonfigurierbaren Logik wird nicht unterstützt. Ebenfalls ist die Anbindung zusätzlicher schneller Schnittstellen nicht möglich. Die Verwendung von Serienerprobten Softwarekomponenten oder ASIC's wird nicht angeboten.

Quelle: [Ric]

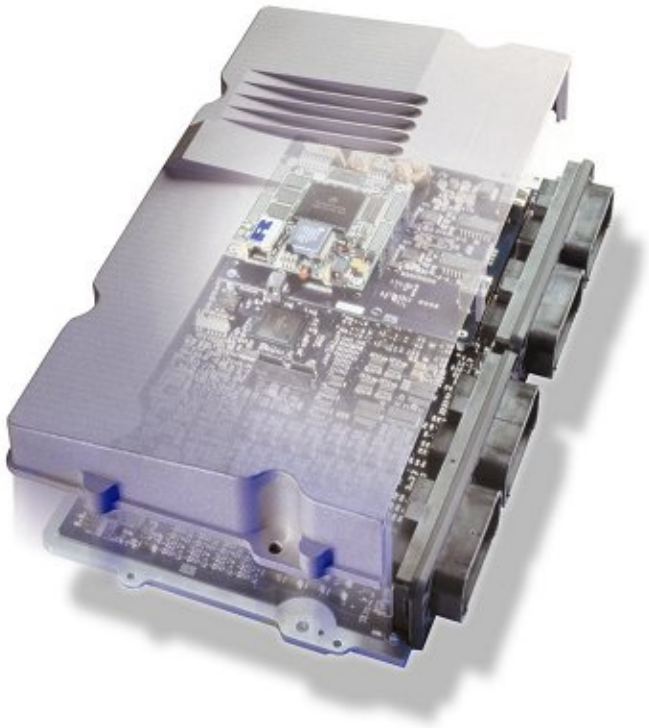


Abbildung 5-14: RICARDO Network Vehicle Controller

Mit keinem der betrachteten Systeme wird die von diesem Ansatz geforderte Verwendung von Serienerprobten Software- und Hardwarekomponenten (ASIC's) unterstützt. Ebenfalls unterstützt keines der Systeme den geforderten massiven Einsatz von rekonfigurierbarer Logik. Daher ist es erforderlich das im folgenden Abschnitt beschriebene Entwicklungssystem zu realisieren.

Kapitel 6

REALISIERUNG AM BEISPIEL EINES DIESEL ENTWICKLUNGS–STEUERGERÄTES

6 Realisierung am Beispiel eines Diesel Entwicklungs–Steuergerätes

Moderne Fahrzeuge bestehen aus einer Vielzahl unterschiedlicher Steuergeräte. Es gibt z.B. Steuergeräte zur Steuerung und Regelung des Verbrennungsmotors, des Getriebes, des Fahrdynamiksystems sowie für eine große Anzahl an Komfortfunktionen wie Klimaregelung, Sitz- oder Spiegelverstellung. In Abbildung 6-1 sind exemplarisch einige Systeme, welche über Steuergeräte verfügen, dargestellt.

Quelle: [Kas02]

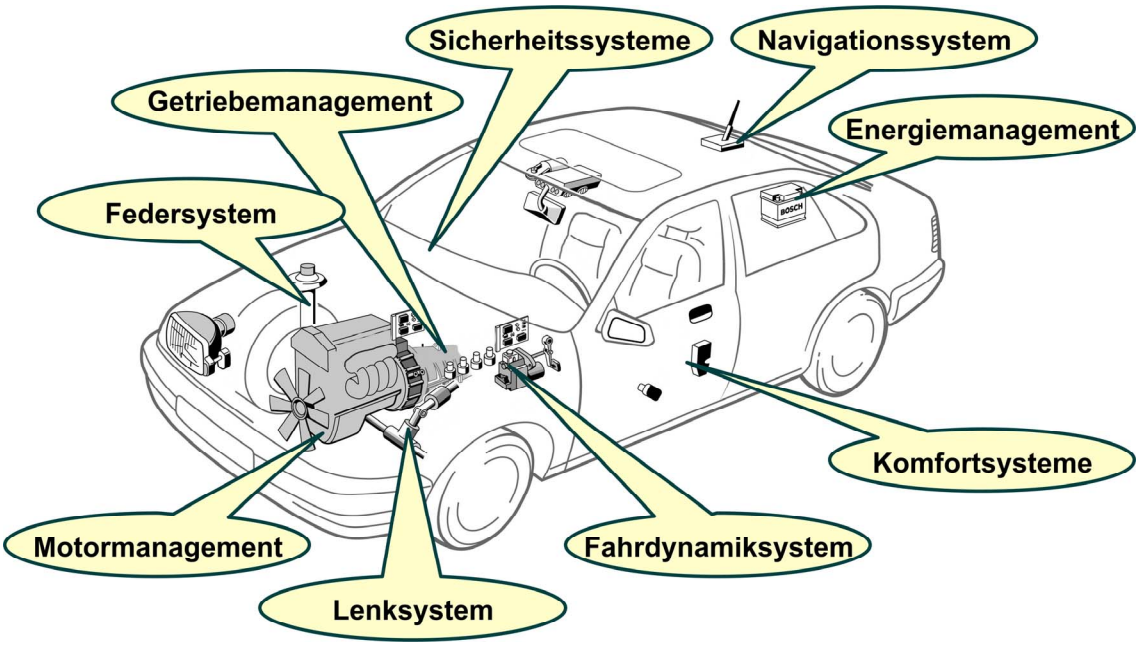


Abbildung 6-1: Systeme moderner Fahrzeuge

Da Motorsteuergeräte bezüglich der Kombination von Rechenleistung und I/O Umfang die größten Anforderungen stellen, eignen sie sich besonders gut als Realisierungsbeispiel für ein universelles Entwicklungs-Steuergerät. Steuergeräte für Verbrennungsmotoren kann man grob in zwei Kategorien unterteilen. Steuergeräte für Ottomotoren und Steuergeräte für Dieselmotoren. In folgendem Beispiel wird exemplarisch die Ansteuerung eines Dieselmotors betrachtet. Dies ist eine beispielhafte Festlegung, welche auf das generelle Prinzip keine Auswirkung hat.

Abbildung 6-2 gibt einen Überblick über die externen Komponenten sowie die benötigten Schnittstellen für die Steuerung bzw. Regelung eines Dieselmotors:

- Auf der linken Seite befinden sich die Eingänge zur Erfassung digitaler und analoger Sensoren des Motors, der Umwelt sowie zur Benutzersteuerung.

- Rechts oben sind die wichtigsten Aktoren abgebildet.

- Rechts unten sind die Kommunikationsschnittstellen zu anderen Steuergeräten, Diagnosesystemen bzw. Visualisierungsschnittstellen zum Fahrer.

Quelle: [Bos02_1]

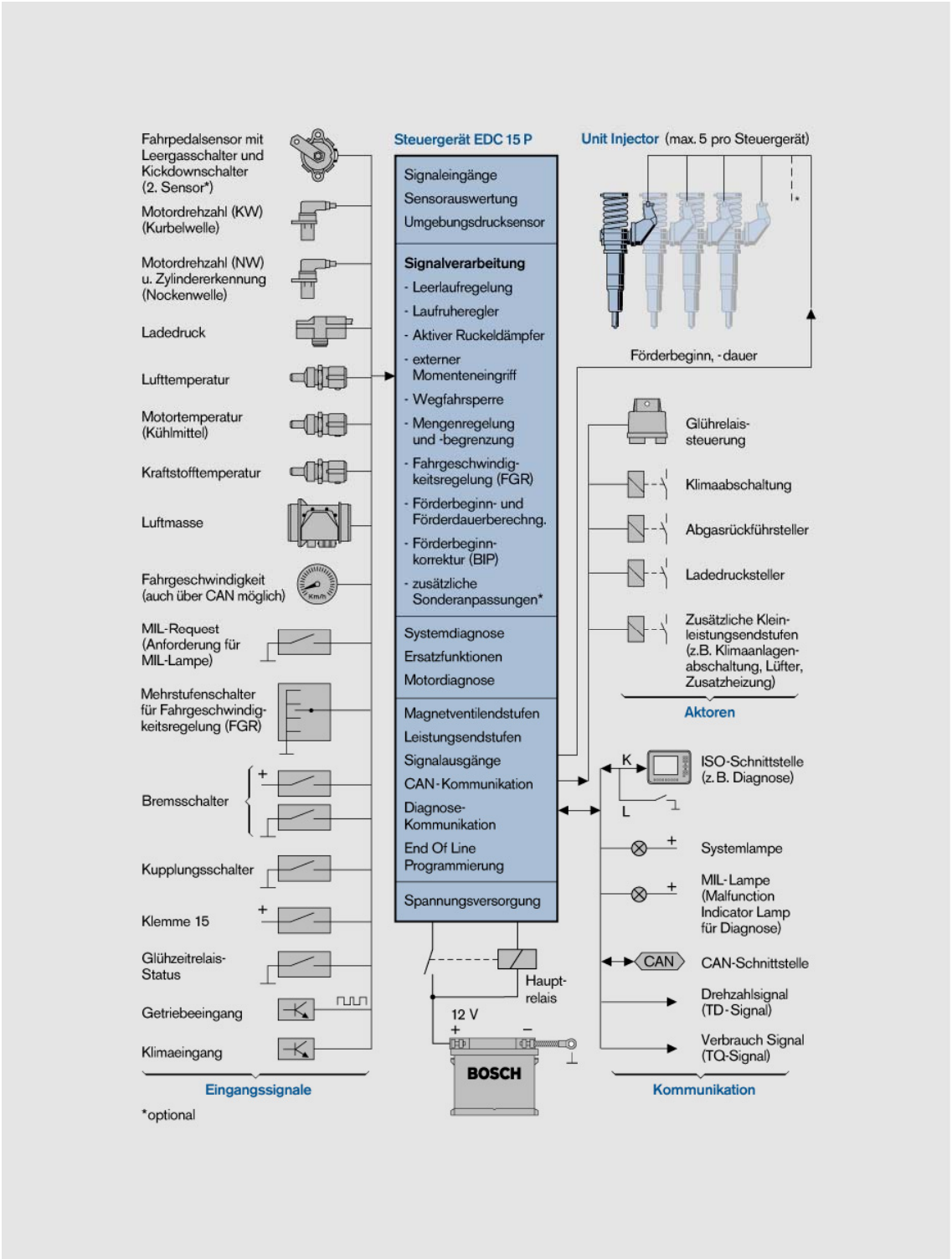


Abbildung 6-2: Schnittstellen Dieselsteuergerät

In den folgenden Abschnitten wird zunächst die Hardware-Architektur und anschließend die Software-Architektur des im Rahmen dieser Arbeit konzipierten und realisierten Entwicklungssystems dargestellt. Zum Abschluss dieses Abschnitts wird exemplarisch anhand einiger wichtiger Funktionen die Anwendung des Systems erläutert.

6.1 Hardware-Architektur des Entwicklungssystems

Abbildung 6-3 zeigt schematisch den Signalfluss des im Rahmen dieser Arbeit konzipierten und realisierten Entwicklungs-Steuergerätes. Die von externen und internen Sensoren generierten Signale werden über die Signalerfassung der Signalverarbeitung zugeführt. Die Aktuatoren werden mit den von der Signalverarbeitung berechneten Ausgangsgrößen über die Signalgenerierung angesteuert.

Bei der Signalerfassung werden zuerst die Sensorsignale, durch die Signal-konditionierungen, auf eine für die Weiterverarbeitung geeignete Größen gebracht. Die analogen Signale werden nach der Signalkonditionierung direkt an den A/D Wandler des Mikrocontrollers geführt. Wohingegen die digitalen Signale über Serien-ASICs oder direkt über die rekonfigurierbare Logik dem Mikrocontroller zur Verfügung gestellt werden. Die Signalauswertung kann von der rekonfigurierbaren Logik, vom Mikrocontroller oder von beiden gemeinsam durchgeführt werden.

Die Signalverarbeitung kann ebenfalls von der rekonfigurierbaren Logik, dem Mikrocontroller oder von beiden gemeinsam durchgeführt werden. Darüber hinaus ist es möglich, Teile oder die komplette Signalverarbeitung an einen leistungsfähigen Simulationsprozessor auszulagern.

Die Signalgenerierung findet im Mikrocontroller, der rekonfigurierbaren Logik oder in Serien-ASIC´s statt. Zur Signalausgabe werden diese Signale bei Bedarf über Leistungs-stellglieder verstärkt und anschließend den Aktuatoren zugeführt.

Funktionen, welche nicht mit den verfügbaren Ressourcen des Entwicklungssystems dargestellt werden können, können auf einem Expansion Board (EB) realisiert werden.

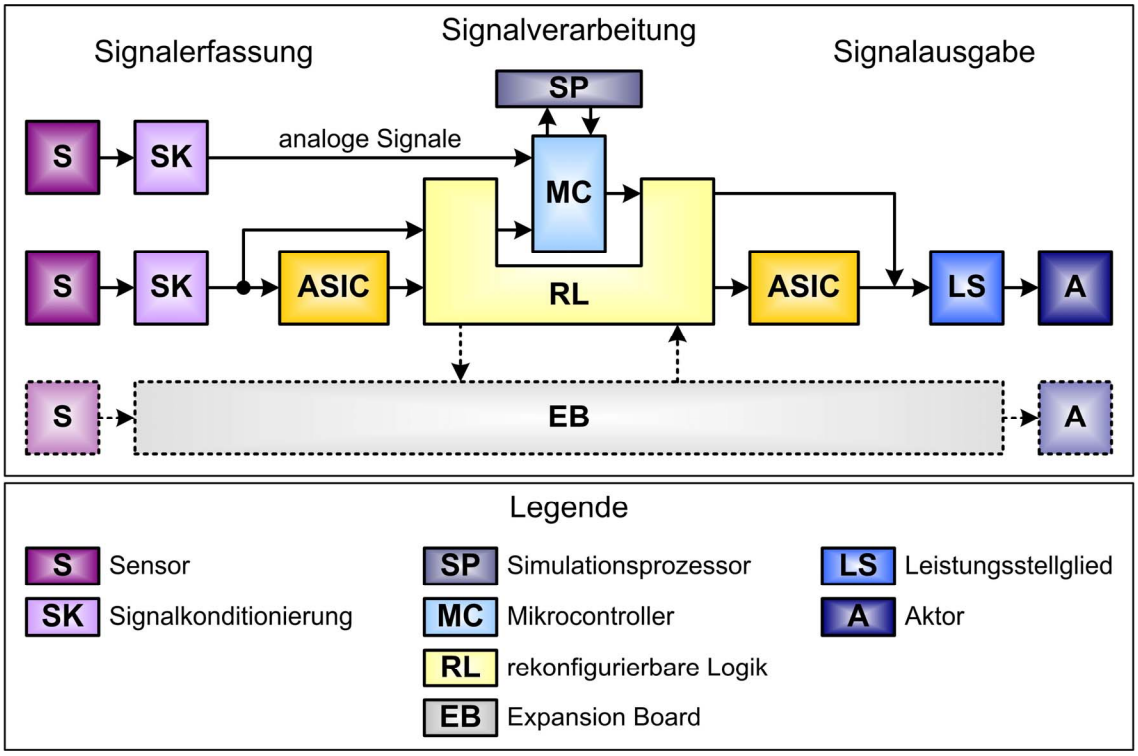


Abbildung 6-3: Signalflussdiagramm Diesel Entwicklungs-Steuergerät

Der durch Partionierung in funktionale Einheiten entstandene interne Aufbau des Entwicklungssystems ist in Abbildung 6-4 dargestellt.

Der Simulationsprozessor (ES1130 / ES1135) kann als optionale Komponente über das Simulationsprozessor Interface (SPI) das System um einen leistungsfähigen Mikroprozessor (MPC750) inkl. RAM und FLASH ergänzen. Das SPI ist als paralleler Multimaster VMEbus realisiert. Der Simulationsprozessor ermöglicht es, mit der großen Rechenleistung und Speicherressourcen leistungsfähige Algorithmen in Fließkommaarithmetik zu berechnen. Auf den Simulationsprozessor sowie das Simulationsprozessor Interface wird nachfolgend nicht näher eingegangen, da hier Standardprodukte der Firma ETAS eingesetzt werden konnten.

Auf dem im Rahmen dieser Arbeit entworfenen und realisierten Mikrocontroller Board (ES1600) sind neben den Schnittstellen zum Simulationsprozessor und zum Adaptationboard der Motorola® Mikrocontroller MPC555 sowie FLASH- und SRAM-Speicher untergebracht.

Auf dem ebenfalls im Rahmen dieser Arbeit entworfenen und realisierten Adaptation Board (PB1640) sind die Signalkonditionierungen für die analogen und digitalen I/O Kanäle sowie die Leistungsendstufen und Serien ASIC's untergebracht. Die rekonfigurierbare Logik (RL) ermöglicht eine flexible Verdrahtung der Mikrocontroller Pins mit den digitalen I/O Kanälen sowie den Leistungsendstufen. Zudem können schnelle Auswerte- und Generierungsalgorithmen integriert werden. Des Weiteren befindet sich auf dem Adaptation Board die für zukünftige Anforderungen offene Erweiterungsschnittstelle (ABI).

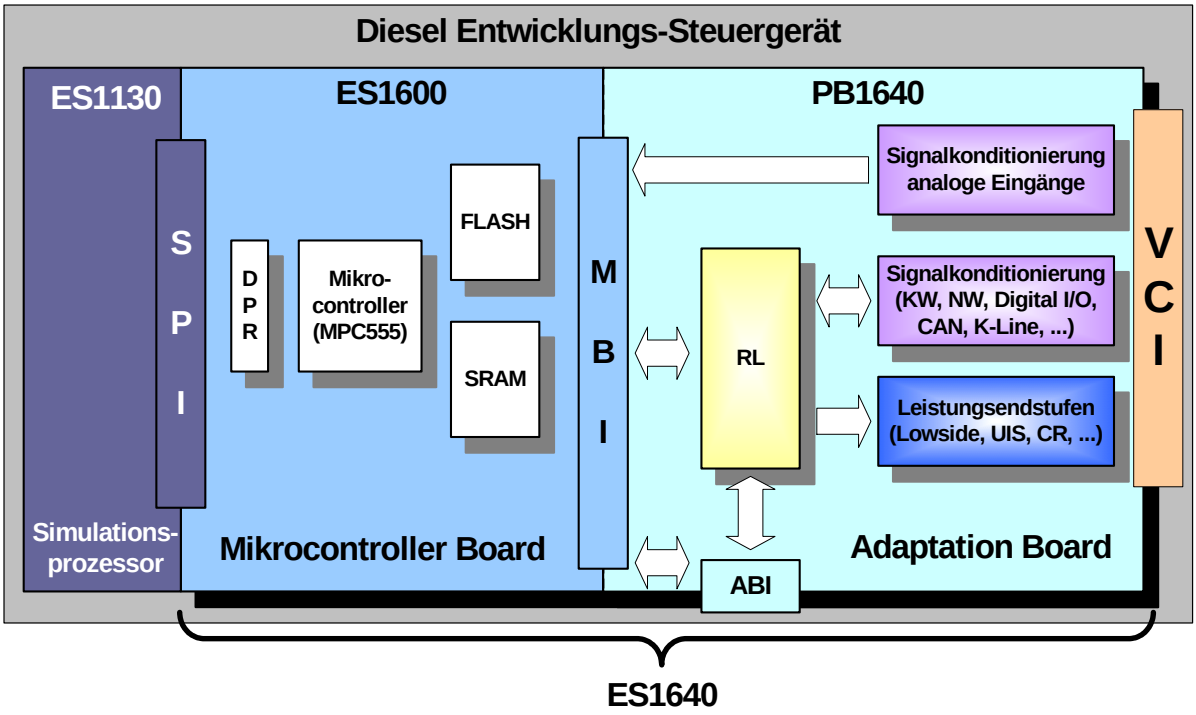


Abbildung 6-4: Blockdiagramm Diesel Entwicklungs-Steuergerät

Die folgenden Abschnitte geben einen Einblick in den Aufbau und die Funktionsweise des Mikrocontroller- und Adaptation Boards.

6.1.1 Mikrocontroller Board (ES1600)

In Abbildung 6-5 ist das Blockdiagramm und in Abbildung 6-6 ein Foto des im Verlauf dieser Arbeit realisierten Mikrocontroller Boards (ES1600) mit den in Abschnitt 5 eingeführten Schnittstellen dargestellt.

Das „**S**imulation **P**rocessor **I**nterface“ (SPI) ist als VME64 Schnittstelle ausgeführt. Im Wesentlichen besteht die Schnittstelle aus zwei 16 Bit breiten DPR's, schnellen und leistungsstarken Bustreibern sowie einer in einem FPGA realisierten Ansteuerung. Über diese Schnittstelle ist es möglich, sehr schnell Daten mit dem Simulationsprozessor auszutauschen.

Über das „**M**ikrocontroller **D**ebug **I**nterface“ (MDI) kann ein Debugger oder ein Applikationssystem angekoppelt werden. Über die Debuggerschnittstelle (BDM) kann ein Programm in das SRAM oder FLASH des Mikrocontrollers geladen und beispielsweise die Ausführung im Einzelschrittbetrieb oder über Breakpoints gesteuert werden. Über die Applikationsschnittstelle (ETK) ist es ebenfalls möglich, ein Programm in das SRAM oder FLASH des Mikrocontrollers zu laden. Zusätzlich können sehr einfach Parameter, Kennlinien und Kennfelder online modifiziert werden, ohne den Mikrocontroller dabei auszubremsen.

Über das „**M**ikrocontroller **B**oard **I**nterface“ (MBI) werden sämtliche Peripherie-Signale des Mikrocontrollers sowie der komplette Adress-, Daten- und Kontroll-Bus auf zwei nach Funktionsgruppen geordneten Steckern zur Verfügung gestellt. An diese Schnittstelle können aufgabenspezifische Adaptation Boards angeschlossen werden.

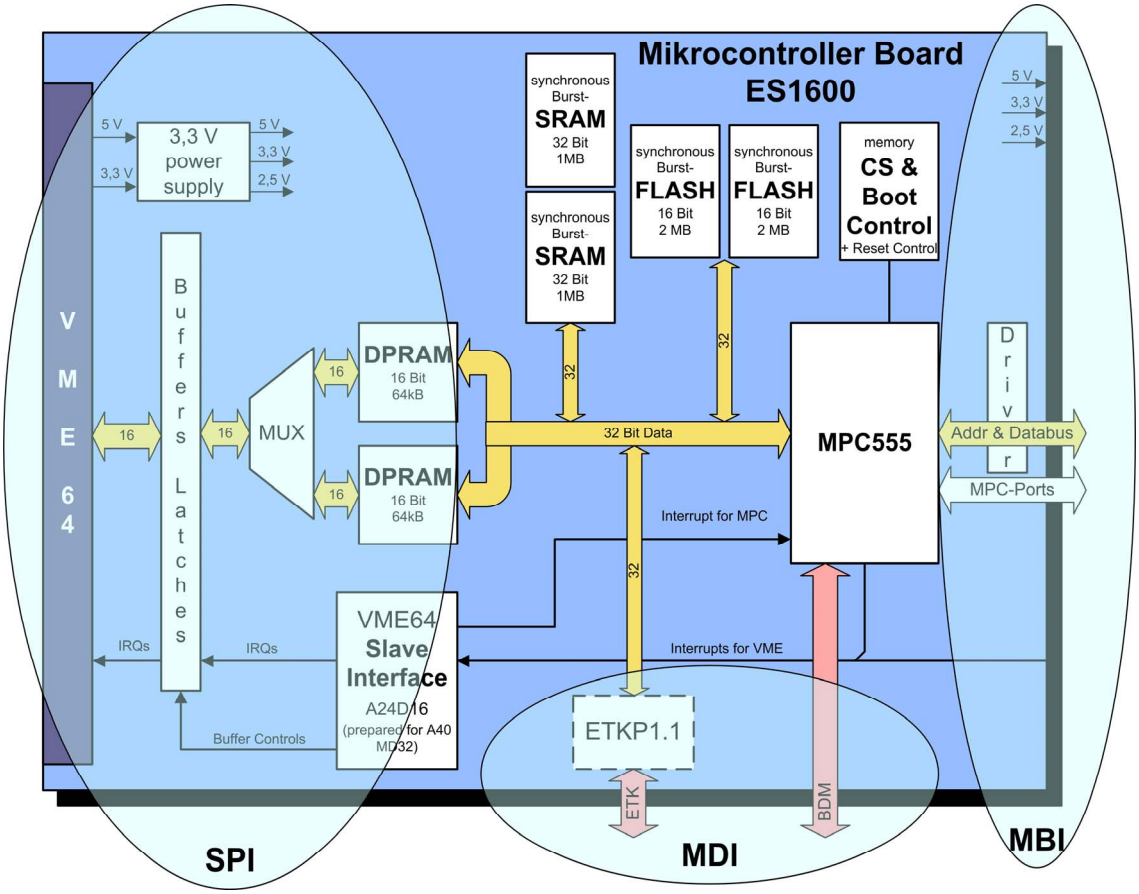


Abbildung 6-5: Blockdiagramm Mikrocontroller Board

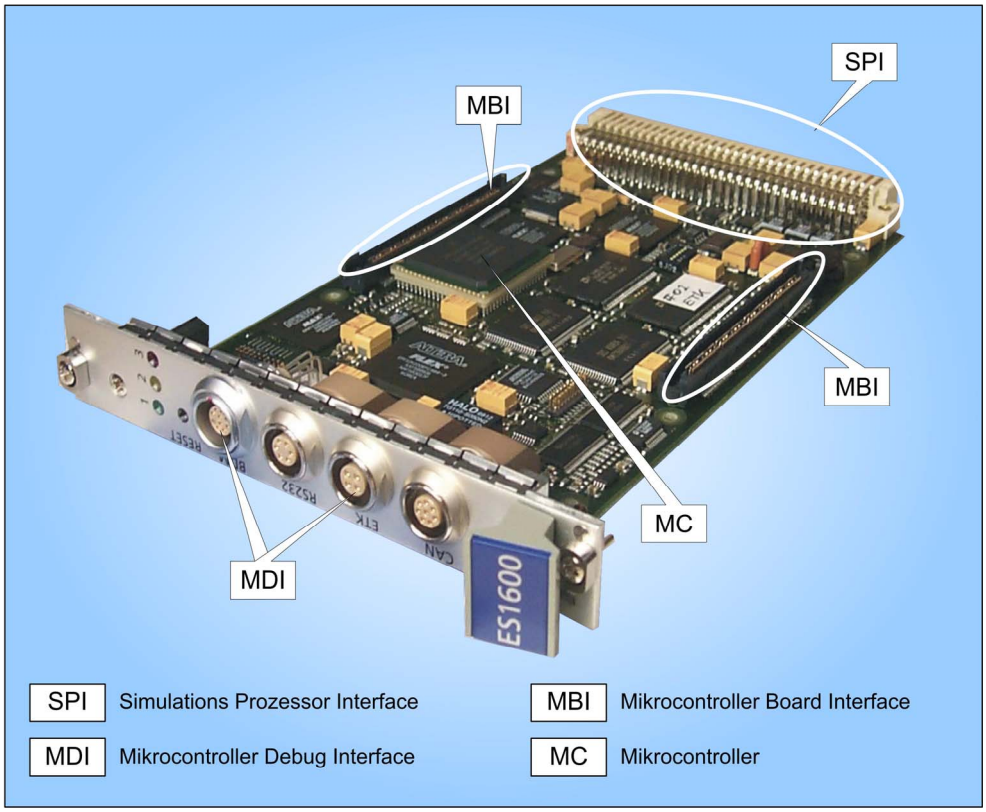


Abbildung 6-6: ES1600

6.1.2 Adaptation Board (PB1640)

Abbildung 6-7 gibt einen Überblick über die wichtigsten Schnittstellen und Funktionsgruppen des Adaptation Boards (PB1640). Das PB1640 wurde im Zuge dieser Arbeit konzipiert und realisiert. Auf die durch Nummern gekennzeichneten Funktionsgruppen wird in den folgenden Unterabschnitten näher eingegangen.

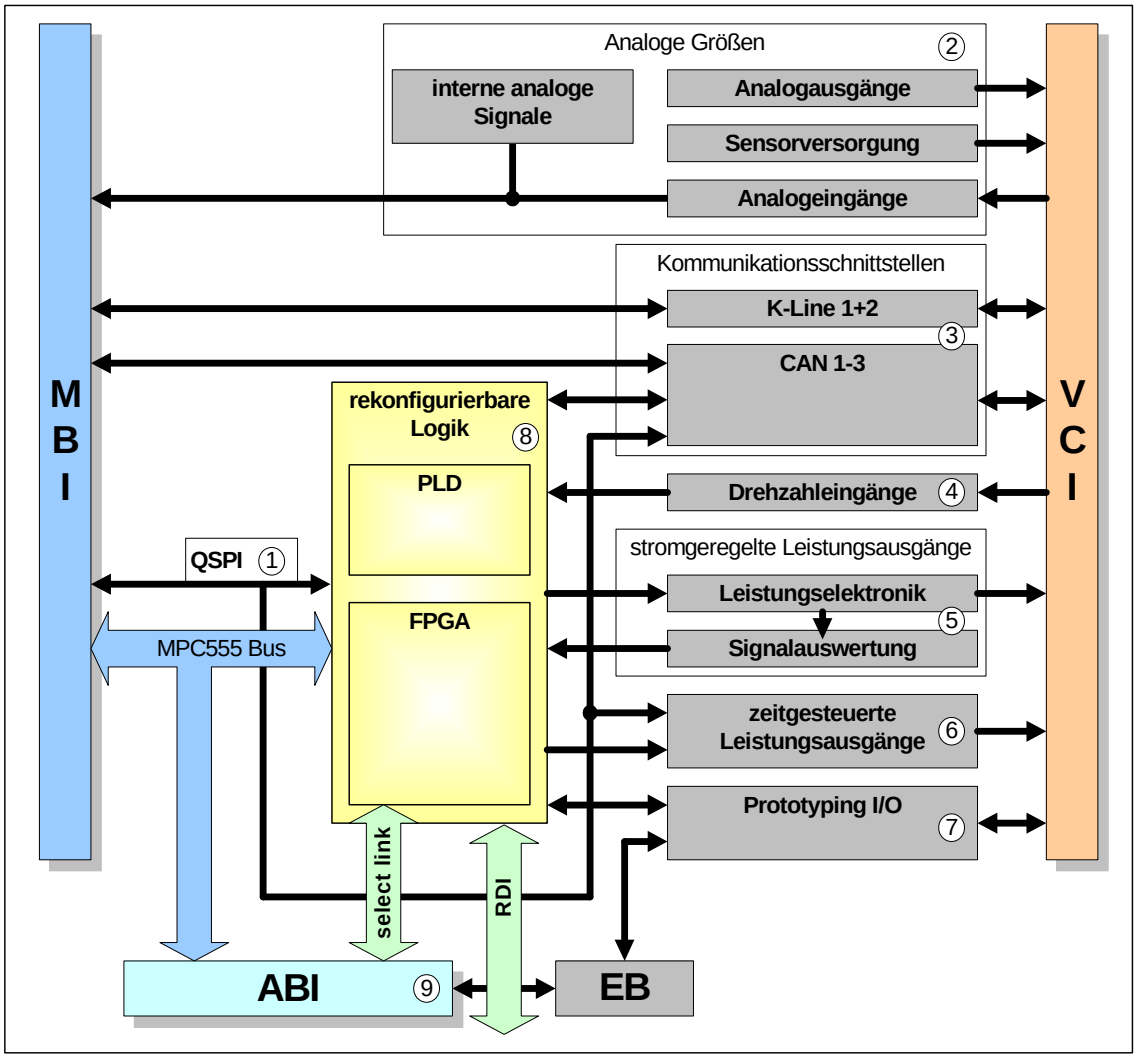


Abbildung 6-7: Blockdiagramm PB1640

Die folgenden Abbildungen zeigen das PB1640 von beiden Seiten. Die unterschiedlichen Schnittstellen sowie die rekonfigurierbare Logik sind zur Orientierung eingezeichnet.

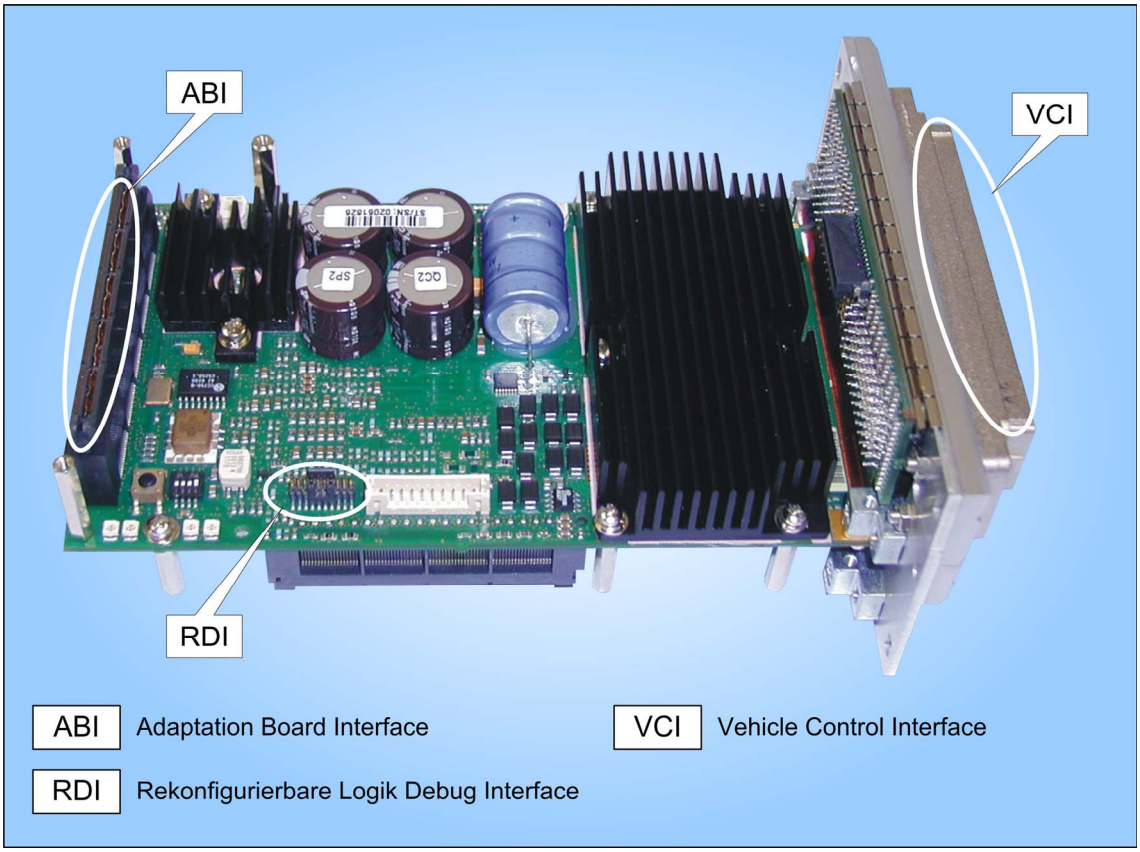


Abbildung 6-8: PB1640 (Ansicht von oben)

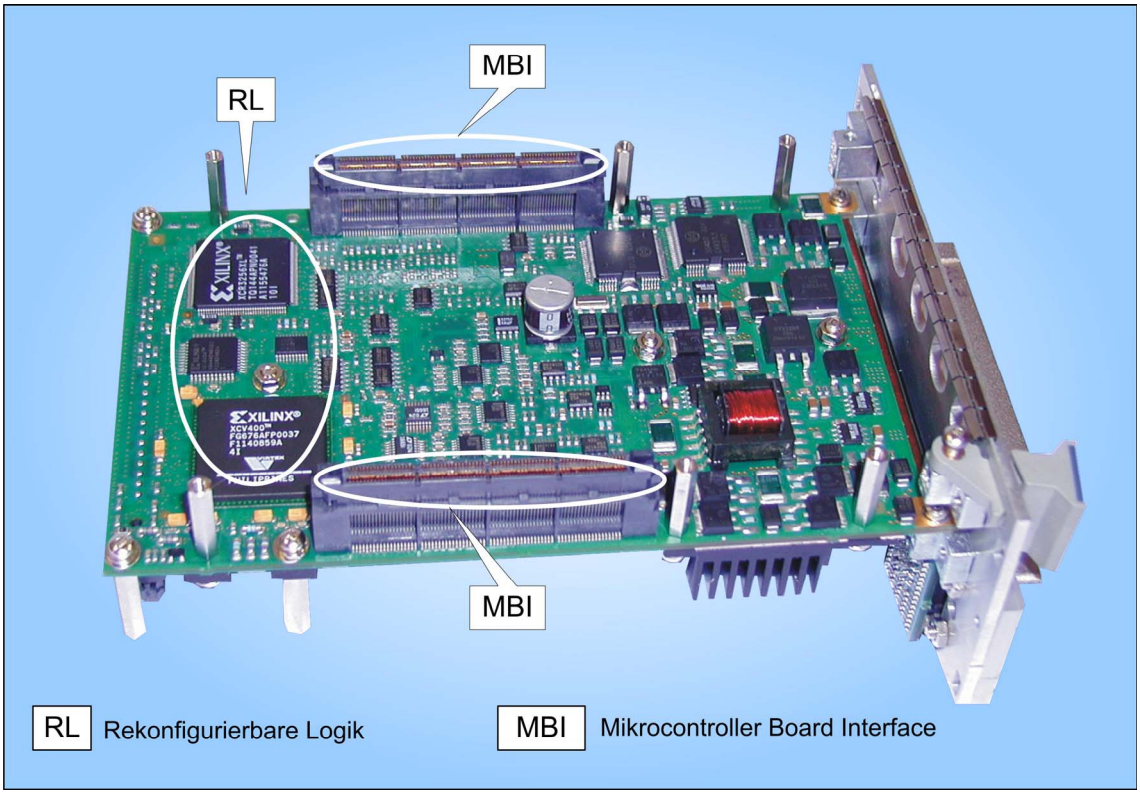


Abbildung 6-9: PB1640 (Ansicht von unten)

6.1.2.1 Interne Daten- und Diagnoseschnittstelle

Um den Verdrahtungsaufwand und somit Kosten und potentielle Fehlerquellen zu minimieren, werden einige Bausteine nicht über eine parallele, sondern über eine standardisierte serielle Schnittstelle angeschlossen. Diese serielle Schnittstelle wird auch bei modernen Bausteinen immer häufiger verwendet, um interne Informationen wie Betriebs- oder Fehlerzustände abzufragen. Diese Zustandsinformationen können unter anderem zu Diagnosezwecken verwendet werden. Bei der intelligenten Serienleistungs-endstufe CJ940 kann beispielsweise abgefragt werden, ob ein Kanal wegen Überstrom abgeschaltet hat.

Eine sehr verbreitete serielle Schnittstelle ist das Queued Serial Peripheral Interface (QSPI). Die QSPI Schnittstelle ist eine Master/Slave Schnittstelle. Der MPC555 der ES1600 ist der QSPI Master. Auf dem PB1640 befinden sich fünf Serien Bausteine, welche über eine QSPI Slave Schnittstelle angesprochen werden können:

- 2 x EEPROM,
- CY310 (Schnittmengenbaustein mit Watchdog, Spannungsregler, CAN-Treiber, etc.),
- CJ940 (Leistungsendstufen),
- CC750 (CAN Controller).

Da der MPC555 nur vier QSPI Chip Selects zur Verfügung stellt, aber fünf Bausteine adressiert werden müssen, werden vom FPGA aus zwei dieser Chip Selects durch Dekodierung drei neue generiert. Siehe dazu auch Abbildung 6-10.

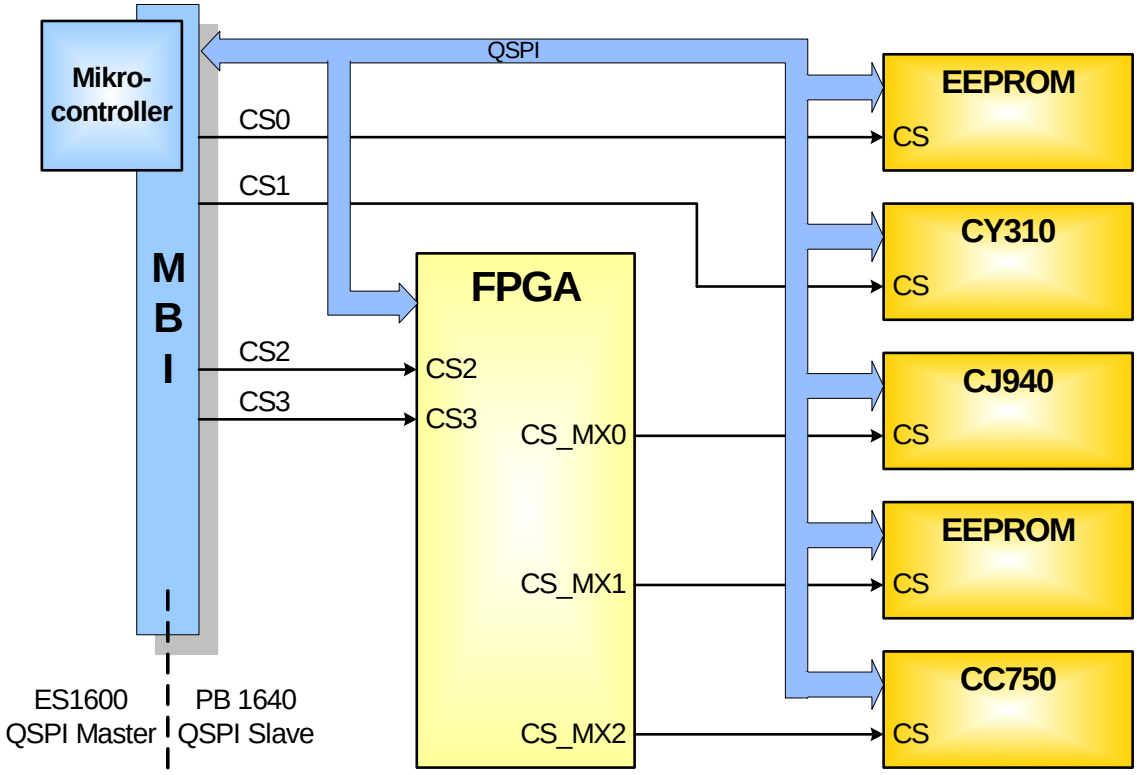


Abbildung 6-10: QSPI Konfiguration

6.1.2.2 Analoge Größen

Der MPC555 besitzt 32 Analogwandlereingänge. Jeweils 16 werden über einen Multiplexer auf einen unabhängigen 10 Bit A/D-Wandler geführt. Alle 32 Analogeingänge werden dem PB1640 über die MBI Schnittstelle zur Verfügung gestellt. Das PB1640 führt 20 dieser Eingänge auf das Vehicle Interface. Mit den verbleibenden 12 werden interne Größen wie Boardspannungen, Luftdruck, Temperatur- und Ströme erfasst. Die Beschaltung der Analogeingänge ist in Abbildung 6-11 dargestellt. Neben den analogen Eingängen werden auch drei Referenzspannungsausgänge zur Versorgung von Sensoren sowie zwei Analog-ausgänge bereitgestellt.

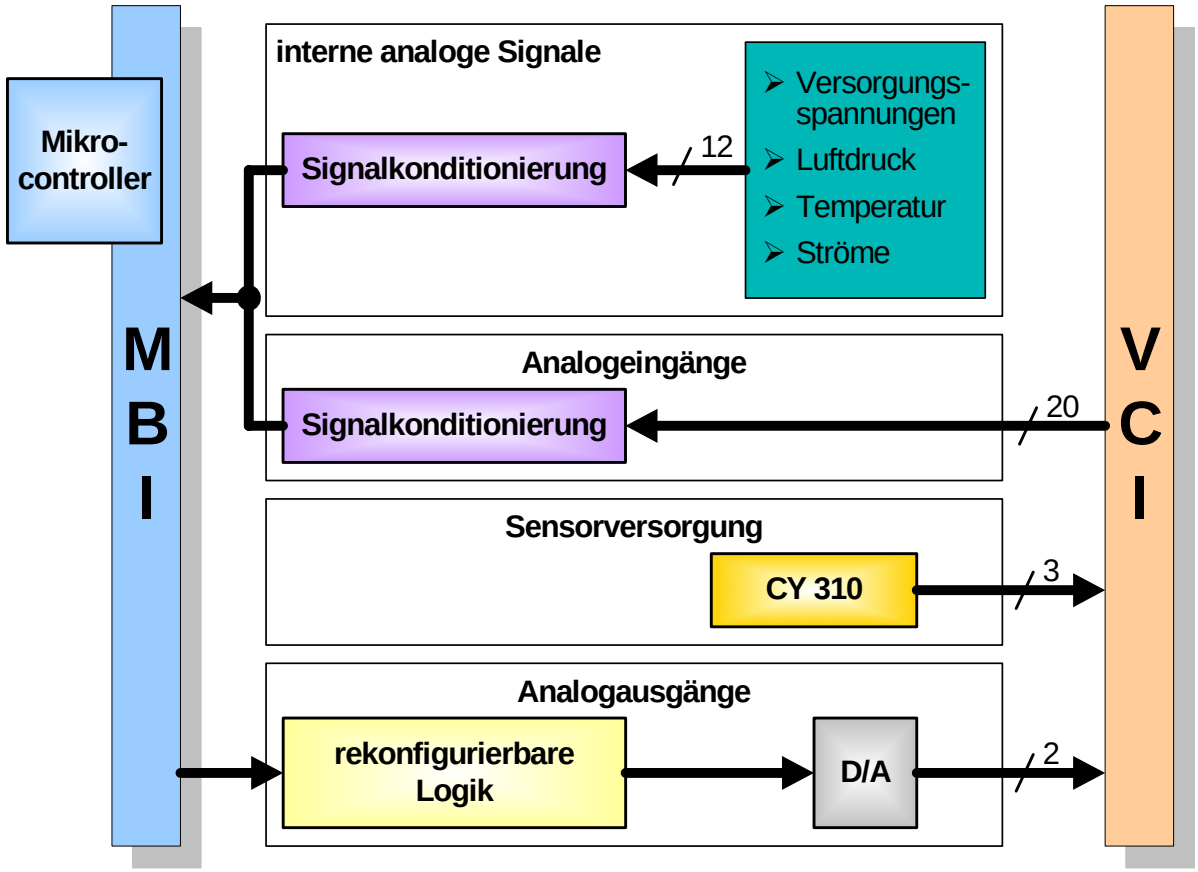


Abbildung 6-11: Generierung und Erfassung analoger Größen

6.1.2.3 Serielle Kommunikationsschnittstellen

Um den Datenaustausch mit weiteren Teilsystemen im Fahrzeug zu ermöglichen, wurden zwei ISO9141 Schnittstellen (K-Line), und drei CAN-Schnittstellen [Ets02] realisiert. Die Kommunikationsschnittstellen sind in Abbildung 6-12 dargestellt.

Um die Identität mit dem Zielsystem herzustellen ist der Physical Layer der K-Line Schnittstellen mit einem ASIC aus der Serie (Schnittmengenbaustein CY310) realisiert. Die K-Line Schnittstelle wird vorwiegend für Diagnosezwecke in Werkstätten und zum „langsamen“ Datenaustausch z.B. mit dem Generator verwendet.

Eine CAN-Schnittstelle (CAN1) wurde als „High Speed“ CAN mit maximal 500 kBit/s realisiert. Als CAN Controller wird einer der beiden im Mikrocontroller vorhandenen verwendet. Der Physical Layer ist ebenfalls mit dem Schnittmengenbaustein CY310 realisiert.

Eine weitere CAN Schnittstelle (CAN2) wurde als „High Speed“ CAN mit maximal 1 Mbit/s realisiert. Der Physical Layer ist bei dieser Schnittstelle nicht direkt mit dem CAN Controller verbunden sondern wird durch das FPGA geführt. Damit ist es möglich, entweder den zweiten CAN Controller des Mikrocontrollers zu verwenden oder alternativ einen spezifischen CAN Controller (z.B. TT-CAN) in der rekonfigurierbaren Logik zu realisieren. Der Physical Layer ist mit dem Serien ASIC CF160 realisiert.

Als Weiteres wurde ein „Fault Tolerant“ CAN (CAN3) realisiert. Als CAN Controller kommt hier der Serien ASIC CC750 zum Einsatz. CAN3 kann alternativ ebenfalls durch das FPGA geführt werden.

Die „High Speed“ CAN Kanäle werden üblicherweise für den Datenaustausch im Antriebsstrang des Fahrzeuges bzw. für Applikationszwecke verwendet. Über den „Fault Tolerant“ CAN können Komfortsysteme integriert werden.

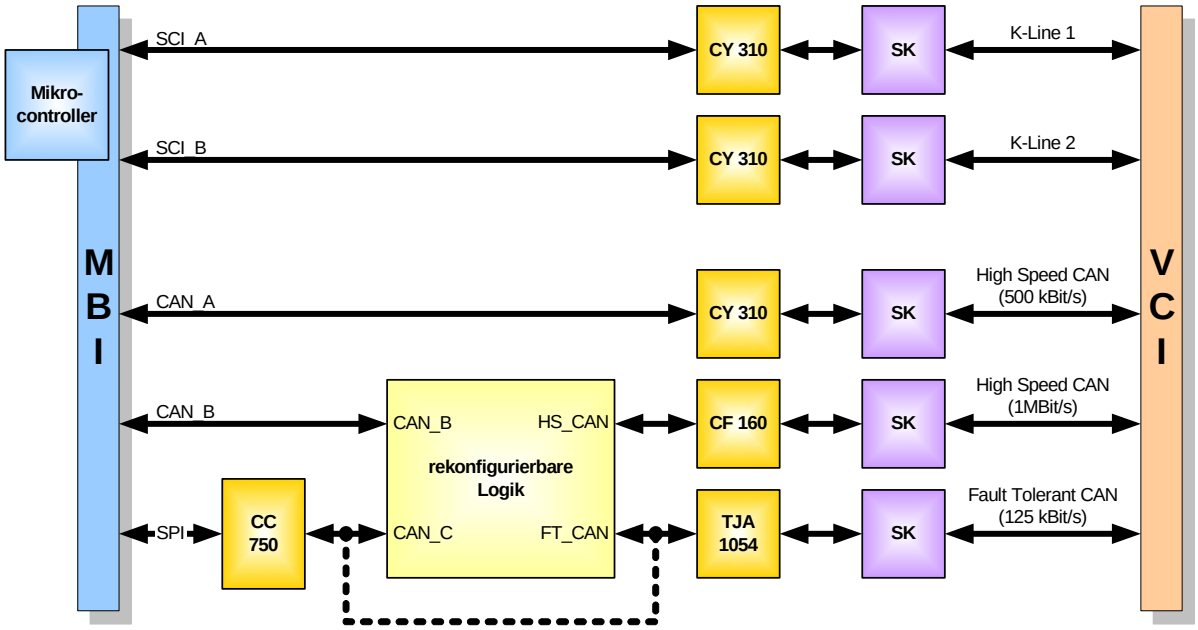


Abbildung 6-12: Serielle Kommunikationsschnittstellen

6.1.2.4 Signalaufbereitung für Drehzahlsignale

Für die Regelung eines Verbrennungsmotors sind die Drehzahl bzw. die Position der Kurbel- und Nockenwelle elementare Eingangsgrößen. Speziell für die heute noch vielfach eingesetzten induktiven Drehzahlsensoren wird eine passende Signalaufbereitung benötigt. Auf dem PB1640 wurde eine sehr flexible Beschaltung realisiert. In Abbildung 6-13 ist die Signalaufbereitung für die Drehzahlsignale dargestellt.

Zukünftige Sensoren werden vermehrt direkt digitale Signale bereitstellen. Diese können dann mit den Prototyping Eingängen (vergl. 6.1.2.7) ausgewertet werden. Weitere Informationen zur Auswertung von Drehzahlsignalen sind im Abschnitt 6.3.1 zu finden.

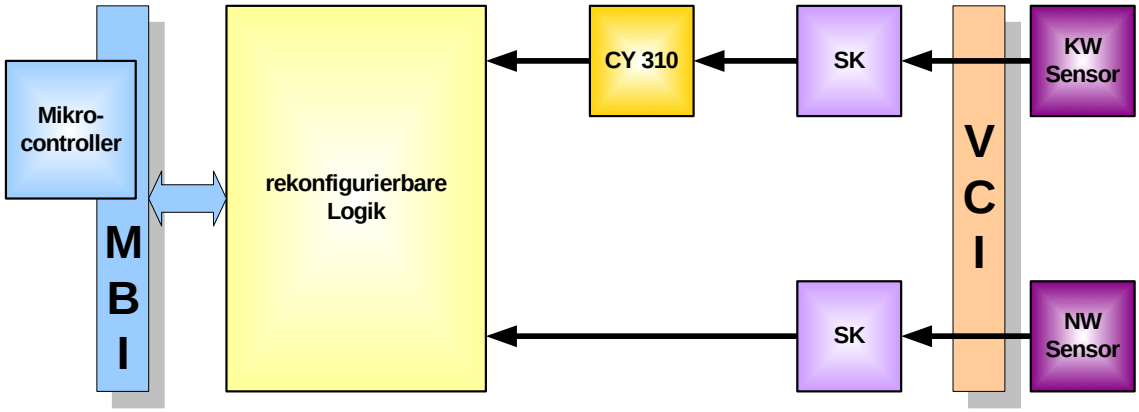


Abbildung 6-13: Signalaufbereitung für Drehzahlsignale

6.1.2.5 Stromgeregelte Leistungsausgänge

Zur sehr schnellen Ansteuerung von Aktoren werden häufig Verfahren mit unterlagertem Stromregler verwendet. Da die Zykluszeiten für eine solche Stromregelung häufig sehr klein sind, eignet sich die rekonfigurierbare Logik sehr gut zur Realisierung dieser Ansteuerung. In Abbildung 6-14 ist die Struktur der für die Stromregelung vorbereiteten Leistungsausgänge dargestellt. Es sind zwei unabhängige Bänke mit jeweils 4 Ausgängen vorhanden. Die Signalauswertung stellt die aufbereiteten Strominformationen der Ansteuerung zur Verfügung. Um die Ansteuerung speziell beim Einschalten zu beschleunigen, ist ein DC/DC Wandler integriert. Dieser liefert ein deutlich höheres Spannungspotential verglichen mit dem im Normalbetrieb verwendeten.

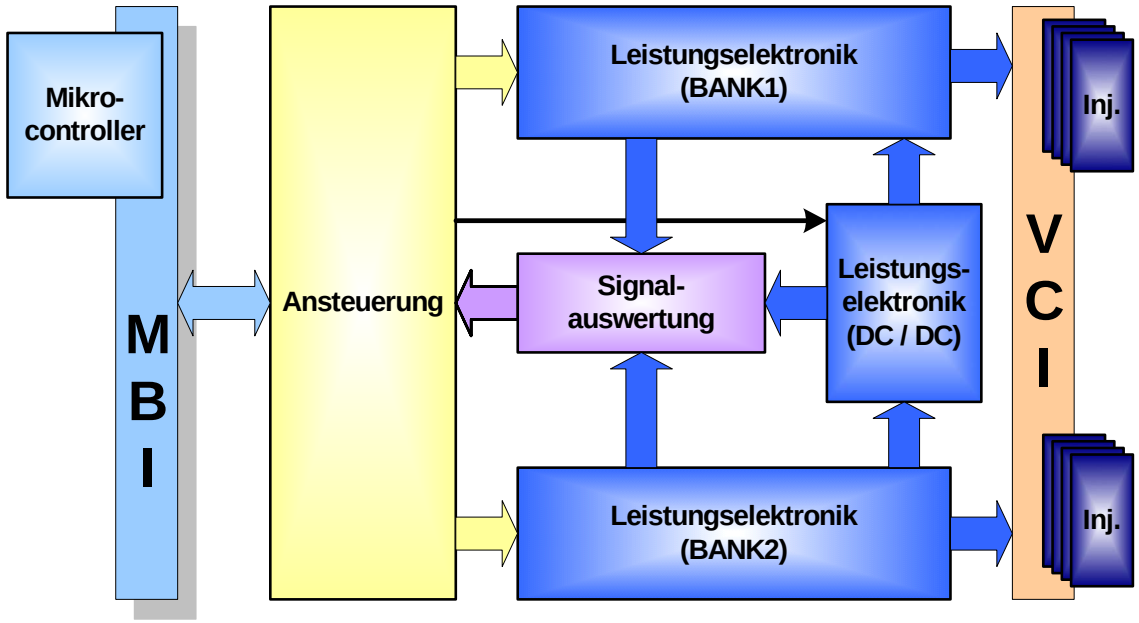


Abbildung 6-14: Übersichtsblockdiagramm Einspritzung

Die Anwendung einer solchen unterlagerten Stromregelung wird in Abschnitt 6.3.2 am Beispiel einer Magnetventilansteuerung beschrieben.

6.1.2.6 Zeitgesteuerte Leistungsausgänge

Für die direkte Ansteuerung von Verbrauchern über Schalt- oder PWM Signale werden Low-Side Endstufen mit unterschiedlicher Spannungs- und Stromfestigkeit zur Verfügung gestellt (Abbildung 6-15). Sämtliche Ausgänge verfügen über Freilaufdioden. Es kommen zwei ASIC's (CJ940, CY310) zum Einsatz, welche auch in Serien-Steuergeräten verwendet werden. Bei einem Ausgang wurde zusätzlich ein Shunt sowie die entsprechende Verstärkerschaltung zur Strommessung integriert. Dieser Strom kann über einen Analogeingang des Mikrocontrollers erfasst und zur Diagnose oder für langsamere Stromregelungen verwendet werden.

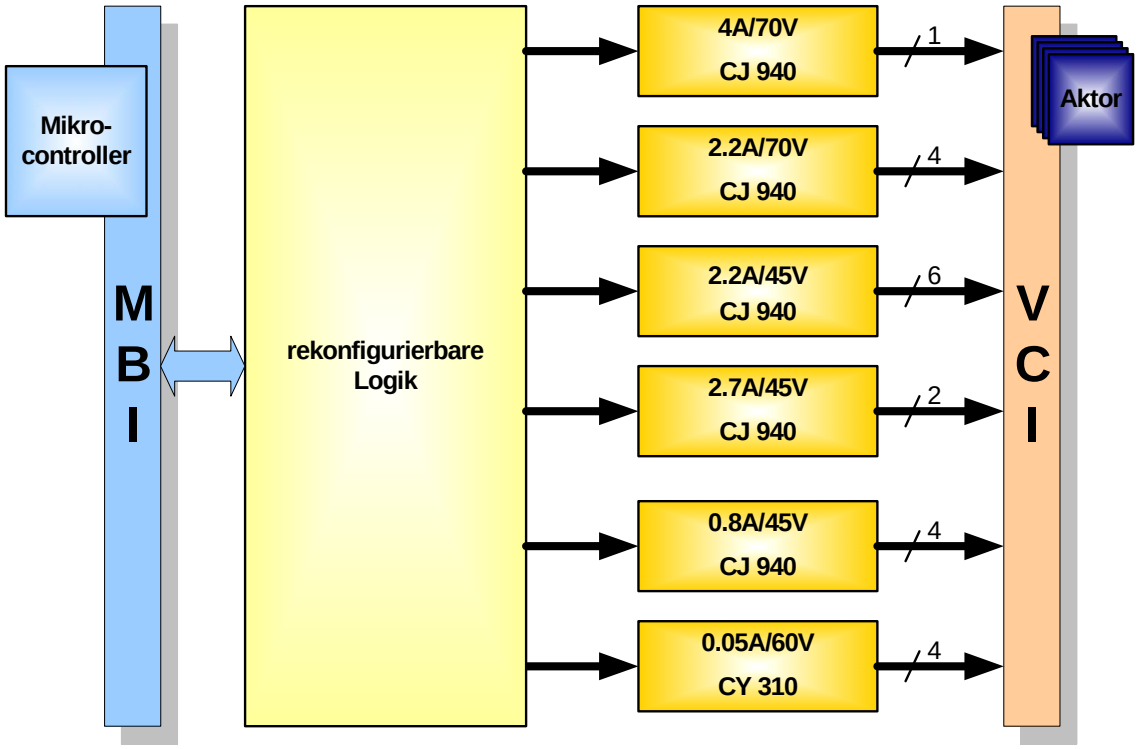


Abbildung 6-15: Leistungsendstufen

6.1.2.7 Prototyping Ein- und Ausgänge

Um über heute bekannte Anforderungen hinaus weitere digitale Signale einlesen und generieren zu können, werden jeweils 16 Eingänge und 16 Ausgänge am Fahrzeugstecker zur Verfügung gestellt. Die Signalkonditionierung dieser Pins ist sehr universell ausgelegt, so dass eine Anpassung an viele Anwendungsfälle möglich ist. Es ist beispielsweise durch entsprechende Bestückung möglich, Spannungsteiler oder Filter zu realisieren. Bei der Defaultbestückung wird die höchste Priorität auf den Schutz der rekonfigurierbaren Logik gelegt. Die Dimensionierung sämtlicher Bauelemente kann projektspezifisch an die geforderten Bedingungen angepasst werden.

Zusätzlich werden 16 weitere Pins am Fahrzeugstecker zur Verfügung gestellt, welche auf einen internen Stecker des PB1640 geführt sind. Über diese Pins können z.B. Signale geführt werden, welche auf einem Expansion Board (EB) generiert werden, welches über das ABI integriert ist.

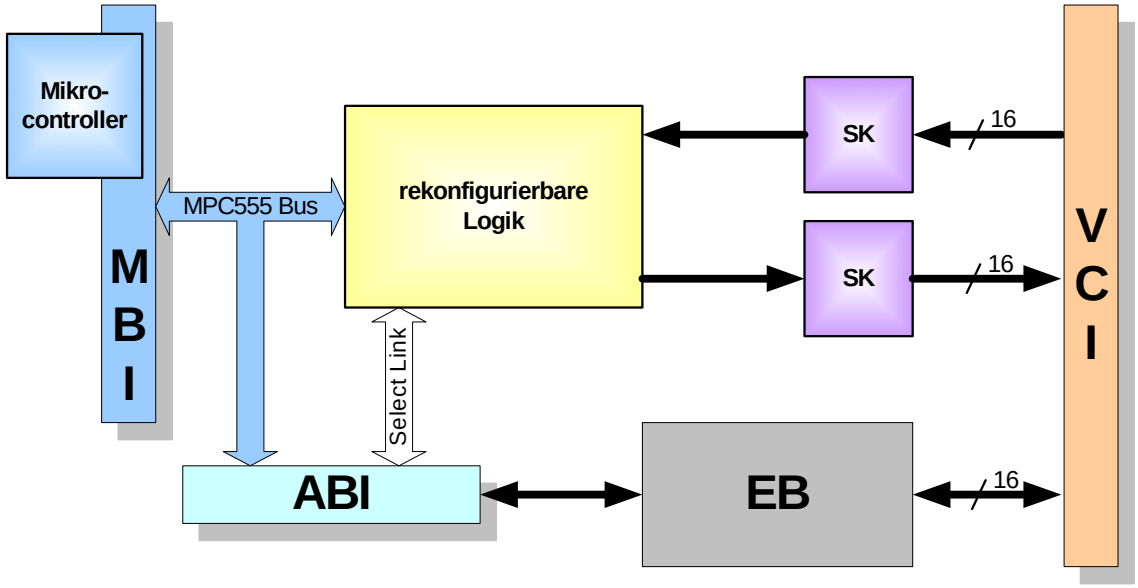


Abbildung 6-16: Prototyping Ein- Ausgänge

6.1.2.8 Erweiterungsschnittstelle (ABI)

Für Anwendungen, bei denen die Kapazität der rekonfigurierbaren Logik auf dem PB1640 nicht ausreicht, oder weitere Interfaces benötigt werden, wurde die Erweiterungsschnittstelle (ABI) definiert. Diese Schnittstelle verfügt neben dem Adress-, Daten und Kontrollbus des MPC555 über eine sehr schnelle und flexible Verbindung zur rekonfigurierbaren Logik des PB1640. Mit dieser Select Link Schnittstelle (Abbildung 6-17) ist es möglich, sehr schnell Daten zwischen FPGAs auszutauschen. Es können Datenraten zwischen 140 und 200 Mbit/Sekunde und Pin realisiert werden. Maximal stehen 34 Pins für jede Richtung zur Verfügung. Physikalisch werden die Signale auf einem 190-poligen Mictor-Stecker zur Verfügung gestellt.

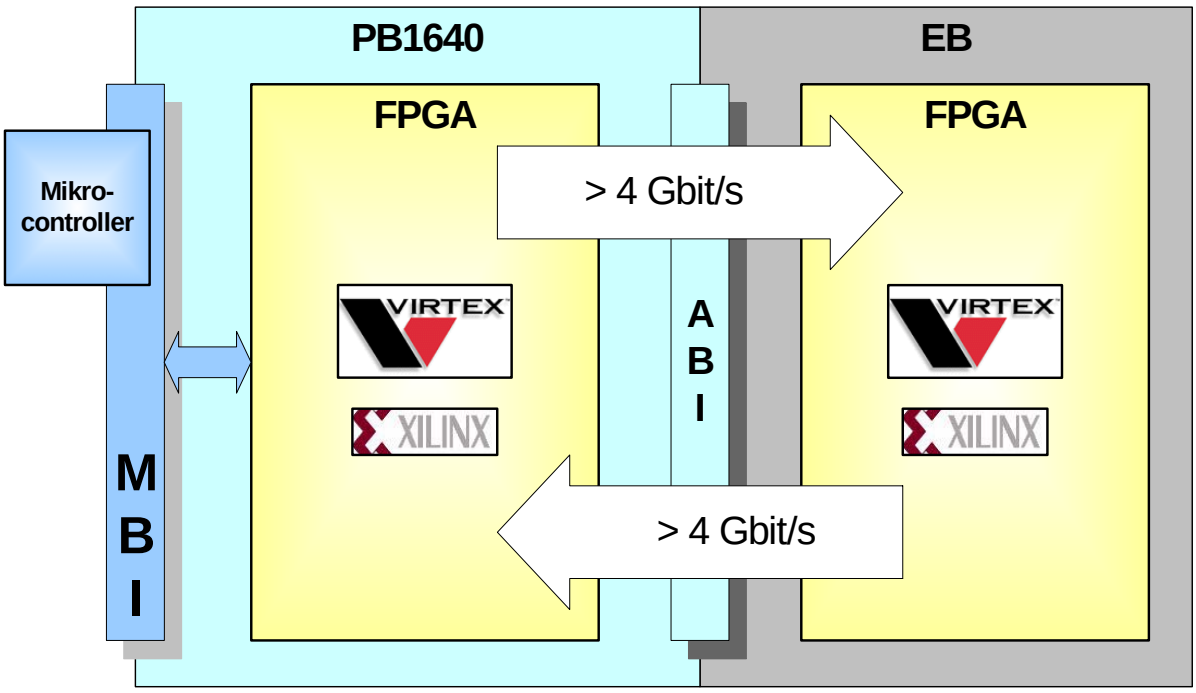


Abbildung 6-17: Select Link Schnittstelle

6.1.2.9 Rekonfigurierbare Logik

Das zentrale Element des PB1640 ist die rekonfigurierbare Logik (RL). Mit der RL ist es sehr einfach möglich, schnelle (in der Größenordnung von µs oder ns) steuerungs- und regelungstechnische Anwendungen direkt in Hardware zu realisieren und somit den Mikrocontroller zu entlasten. Neben den exemplarisch implementierten Anwendungen Drehzahlerfassung und Einspritzung können weitere projektspezifische Anwendungen sehr einfach dargestellt werden. Für Anwendungen, welche die Leistungsfähigkeit der auf dem PB1640 integrierten RL übersteigen, wurde die Erweiterungsschnittstelle (ABI) eingeführt.

Wie in Abbildung 6-18 dargestellt, setzt sich die RL aus einem programmierbareren Logik-Baustein CPLD und einem Field Programmable Gate Array FPGA zusammen. Diese Kombination wurde gewählt, um die Vorzüge beider Architekturen nutzen zu können. Beide Bausteine sind über das MBI mit dem Mikrocontroller verbunden. Der Mikrocontroller kann über den Adress-, Daten und Kontrollbus sehr schnell (in der Größenordnung von ns) auf das CPLD und den FPGA zugreifen. Das FPGA hat zusätzlich noch eine sehr schnelle und flexible Verbindung (Select Link) zum „**A**daptation **B**oard **I**nterface“ (ABI).

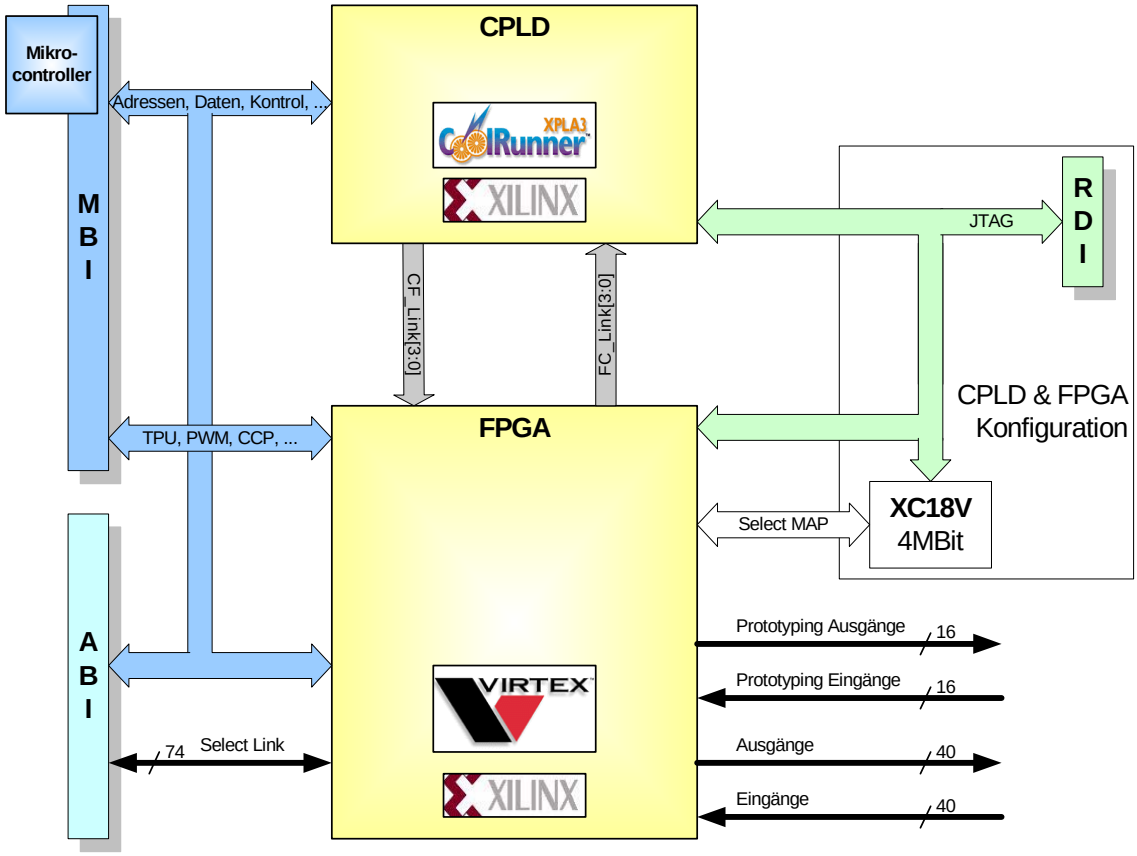


Abbildung 6-18: Rekonfigurierbare Logik

Als CPLD wird ein Baustein der CoolRunner [Xil02] Familie der Firma **XILINX®** verwendet. Die Funktion, welche dieser Baustein erfüllen soll, wird über eine serielle JTAG Schnittstelle im Baustein abgelegt. Einmal programmiert bleibt diese Information bis zur nächsten Umprogrammierung in einem nichtflüchtigen Speicher im Baustein erhalten. Beim Einschalten des Systems ist der CPLD somit sofort funktionsbereit und kann die Initialisierung der restlichen Bausteine- sowie die Freigabe der Resets und der Endstufen kontrollieren.

Als FPGA wird ein SRAM basierter VIRTEX™ [Xil01] Baustein ebenfalls der Firma **XILINX®** verwendet. Die Funktion, welche dieser Baustein erfüllen soll, kann nicht dauerhaft im

Baustein abgelegt werden, da er SRAM basiert ist und somit bei jedem Spannungsausfall seine Konfiguration verliert. Zur dauerhaften Speicherung der Konfiguration wird ein separater nichtflüchtiger Speicherbaustein verwendet. Dieser Speicherbaustein kann, wie das CPLD, über die JTAG Schnittstelle konfiguriert werden. Beim Einschalten wird die Konfiguration von dem Speicherbaustein über eine spezielle parallele Schnittstelle (Select MAP) in den FPGA geladen. Der FPGA selbst ist dennoch mit der JTAG Schnittstelle verbunden. Für Testzwecke oder für eine teilweise oder vollständige Rekonfiguration des Bausteins im Betrieb kann direkt die JTAG Schnittstelle verwendet werden, ohne den Speicherbaustein umprogrammieren zu müssen. Bei der Festlegung der Größe bzw. Komplexität des FPGA wurde ein Kompromiss zwischen Baugröße und Flexibilität gesucht. Die Wahl ist auf einen Baustein mit 400.000 Gattern und 676 Pins gefallen.

Um auf die JTAG Schnittstelle zugreifen zu können, wurden zwei alternative Möglichkeiten implementiert. Zum einen wurde ein Programmierstecker integriert, über den von einem externen Programmieradapter auf die JTAG Schnittstelle zugegriffen werden kann. Damit ist es möglich, mit einer von **XILINX®** zur Verfügung gestellten Software (IMPACT) den Baustein zu programmieren. Des Weiteren wurde die JTAG Chain über eine Umschaltlogik an Port Pins des Mikrocontrollers angebunden. Damit ist es möglich, ohne zusätzlichen Programmieradapter alle JTAG Bausteine zu programmieren. Wie in Abbildung 6-19 zu sehen ist, sind die drei JTAG fähigen Bausteine (Konfigurationsspeicher, FPGA und CPLD) in einer Kette hintereinander geschaltet, so dass über eine einzige Schnittstelle alle drei Bausteine angesprochen werden können.

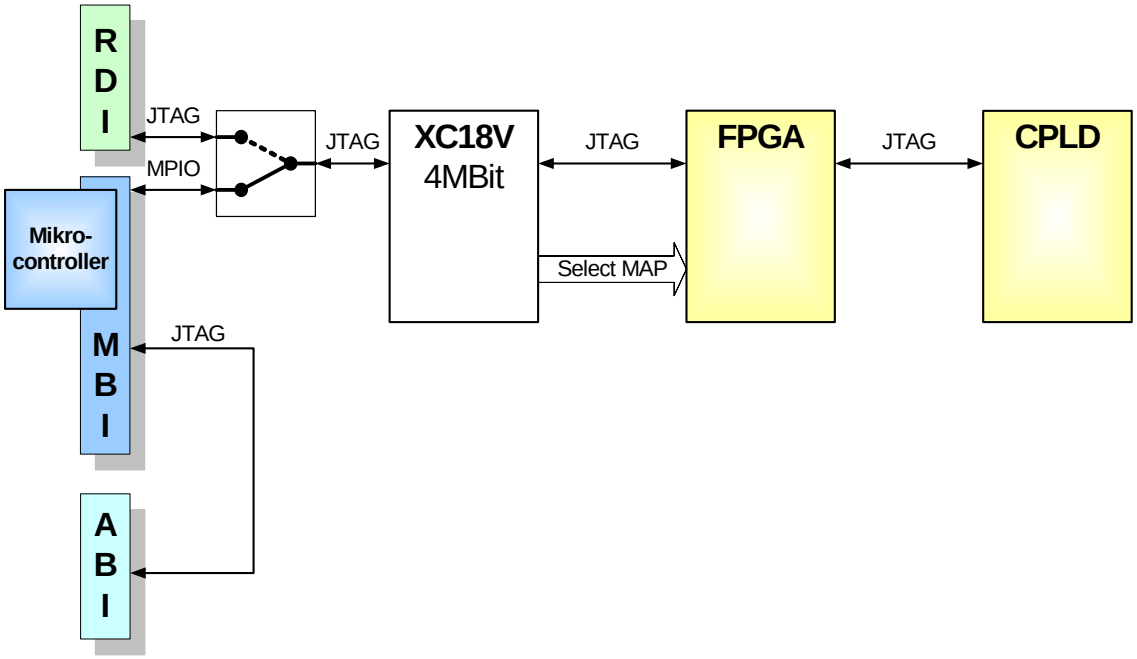


Abbildung 6-19: JTAG Chain der ES1640

Die externe JTAG Schnittstelle wird über einen Programmieradapter an den PC angeschlossen. Auf dem PC kann die IMPACT Software von **XILINX®** (siehe Abbildung 6-20) verwendet werden.

Diese Software kann direkt die von der **XILINX®** Entwicklungsumgebung generierten Konfigurationsfiles lesen und in die unterschiedlichen Bausteine programmieren. Zusätzlich kann zur Fehlersuche im FPGA das ChipScope von **XILINX®** eingesetzt werden. Damit ist es möglich, über die JTAG Schnittstelle FPGA-interne Signale aufzuzeichnen und über den PC zu visualisieren.

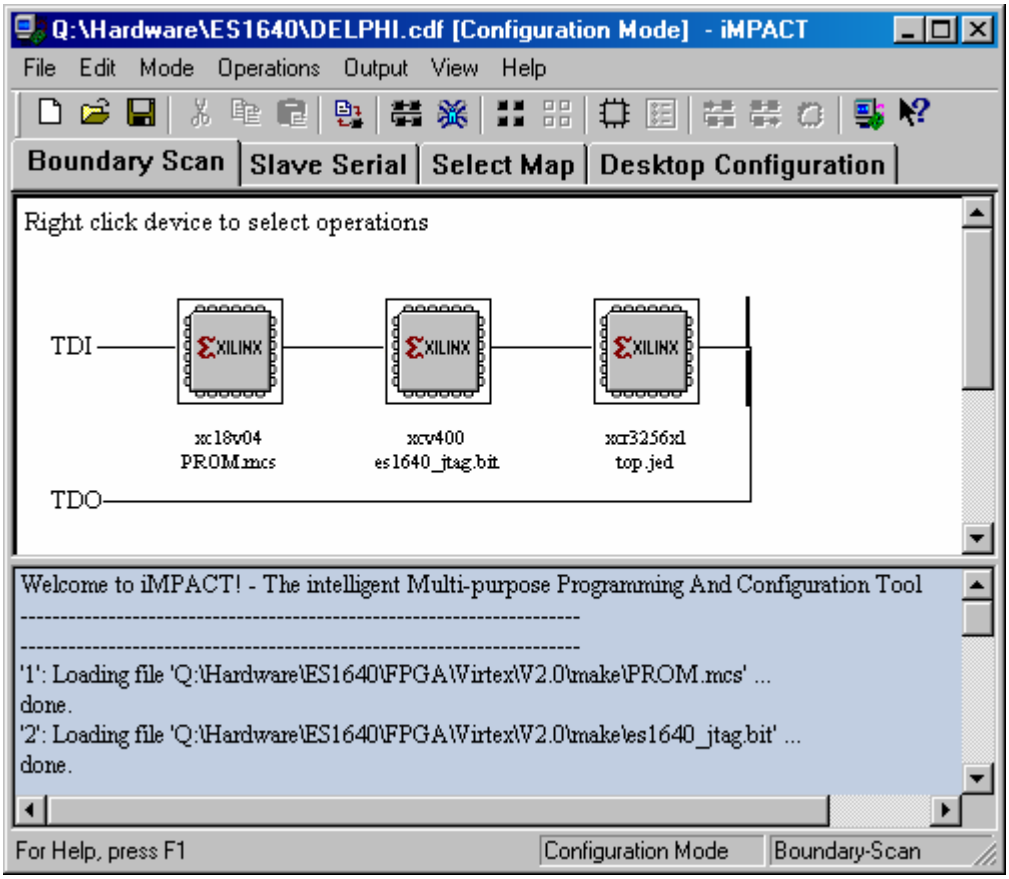


Abbildung 6-20: IMPACT Benutzeroberfläche

Die über die IMPACT Software dargestellte Möglichkeit, die rekonfigurierbaren Bausteine zu programmieren, erfordern spezielle Hardwarekomponenten (Programmieradapter). Das ist für die initiale Programmierung im Labor anwendbar. Soll aber ein Update im Feld bzw. zur Laufzeit durchgeführt werden, wird eine Lösung ohne zusätzliche Hardware-komponenten angestrebt.

Ein Lösungsansatz, unter anderem zum Programmieren der JTAG Bausteine (Konfigurationsspeicher, FPGA und CPLD), wurde in [Kar03] erarbeitet. Darin wird ein vorhandenes Hardware Service Pack (HSP) um die benötigte Funktionalität des Zugriffs auf Bausteine von **XILINX®** erweitert. Mit dem erweiterten HSP ist es möglich, vom PC über eine Ethernet Schnittstelle Daten zum Systemcontroller des VMEbus (ES1120) und von dort über den VMEbus zum DPR der ES1640 zu übertragen. Über diesen Weg ist es dann möglich, beliebige Software auf dem Mikrocontroller der ES1640 ablaufen zu lassen. Mit dieser Software können dann die Logik Bausteine programmiert werden.

6.1.2.9.1 Rekonfigurierbares Erweiterungsmodul

Als eine mögliche Ergänzung des PB1640 kann das ebenfalls im Rahmen dieser Arbeit entworfene und realisierte FPGA3 Modul verwendet werden. Das Expansion Board FPGA3 kann direkt über die ABI Schnittstelle mit dem PB1640 verbunden werden. Es besteht, ähnlich wie das PB1640, aus einem CoolRunner CPLD und einem FPGA. Bei dem FPGA handelt es sich um einen VIRTEX™-E Baustein der Firma **XILINX®** mit 1.600.000 Gattern. Zusätzlich verfügt das FPGA3 Modul noch über SRAM (256k x 16) und FLASH (512k x 16). Die folgenden Abbildungen zeigen das FPGA3 Modul von beiden Seiten.

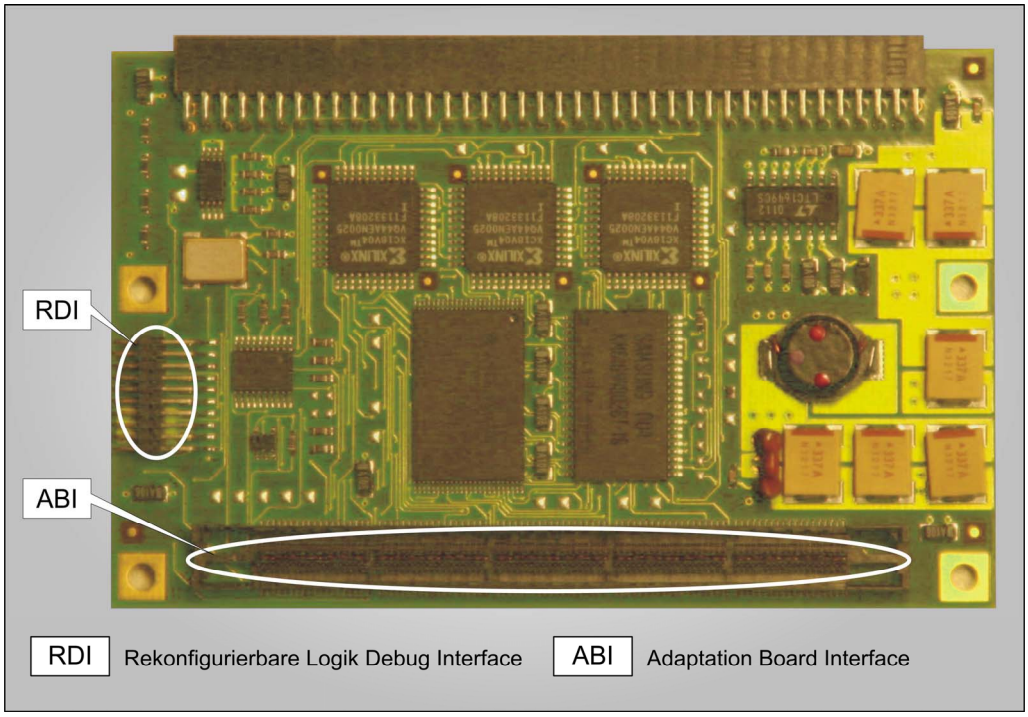


Abbildung 6-21: FPGA3 unten

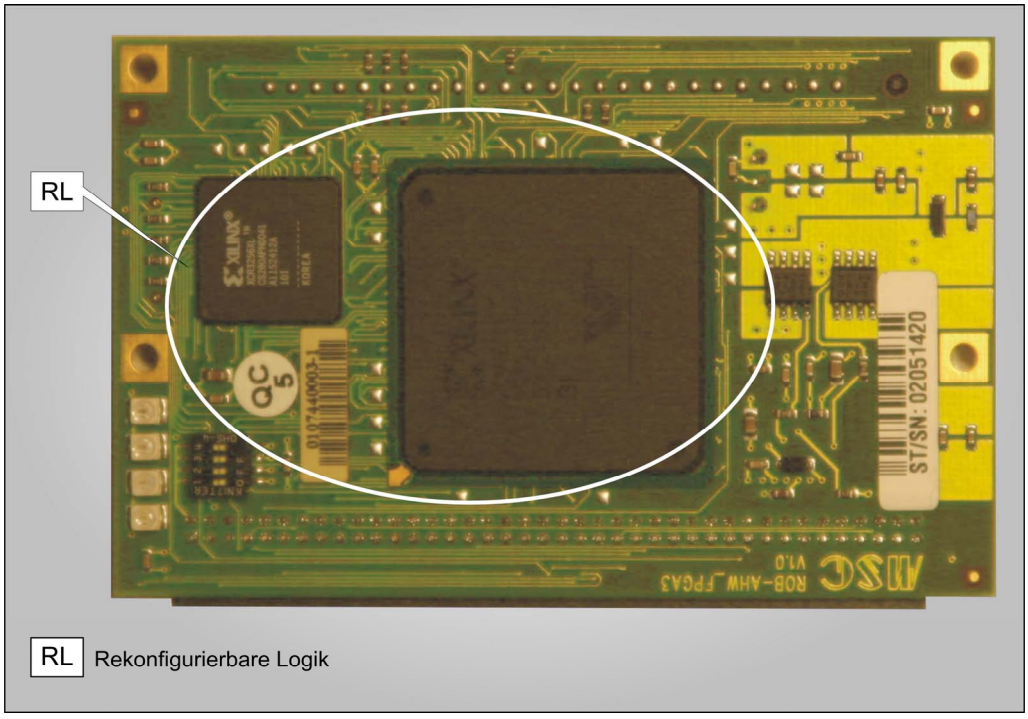


Abbildung 6-22: FPGA3 oben

6.1.2.9.2 Hardware- und Software-Codesign

Die Hardware/Software (HW/SW) Partitionierung, also die Entscheidung, welche Funktionalität in HW und welche in SW abgebildet wird, wird meist aus der Erfahrung heraus getroffen. Nach abgeschlossener Partitionierung startet die HW- und SW-Entwicklung in getrennten Teams. Jedes Team testet zuerst seine eigenen Ergebnisse. Fehlerfreiheit und Akzeptanz des Systemverhaltens kann mit hinreichender Sicherheit allerdings erst nach der Systemintegration festgestellt werden. Aktuell ist die System-integration erst möglich, wenn sowohl die HW, als auch die SW zur Verfügung stehen. Das ist beim Einsatz von ASIC's jedoch erst nach vollständig abgeschlossener HW-Entwicklung zuzüglich der zeitaufwändigen ASIC Erstfertigung möglich.

Kombiniert man nun das zielsystemidentische Rapid Prototyping mit Ansätzen des HW/SW-Codesigns, so ist man direkt nach der Systemspezifikation und einer ersten vorläufigen Partionierung in der Lage, sowohl die Hardware als auch die Software zu implementieren. Der Aufwand für die Hardwareimplementierung beschränkt sich dabei vorwiegend auf das FPGA. Damit ist es nach relativ kurzer Zeit möglich, erste Tests durchzuführen. Anhand der Testergebnisse kann dann in einer oder in mehreren Iterationen die Partionierung angepasst und die Implementierung der Software und Hardware optimiert werden. Sobald man mit dem Systemverhalten und dem Ressourcenverbrauch zufrieden ist, kann die Erstellung der finalen Hardware und Software angegangen werden. Bei der Hardware kann das beispielsweise ein ASIC sein.

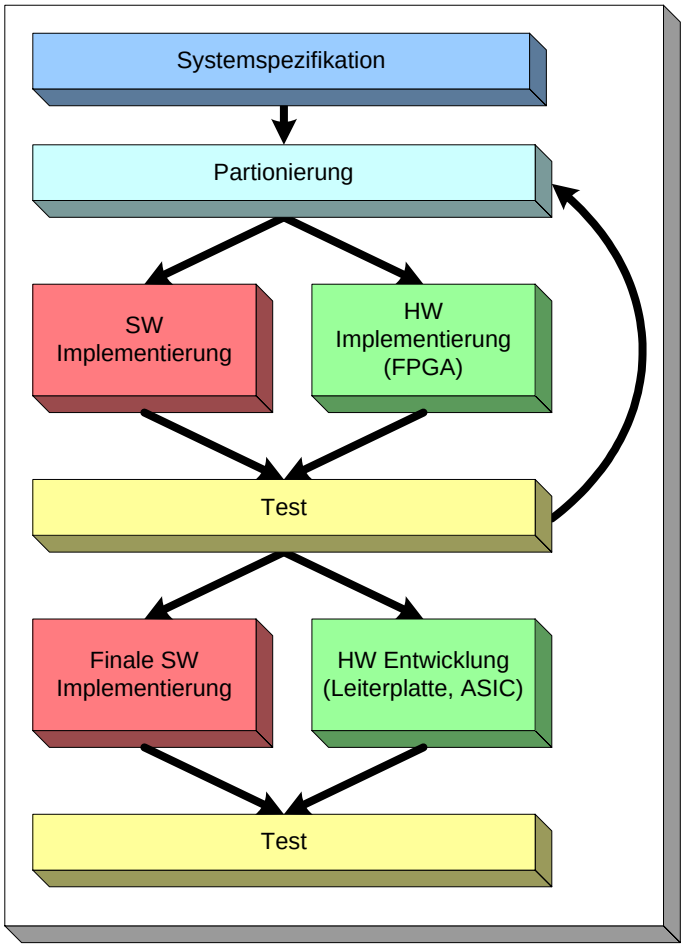


Abbildung 6-23: HW/SW-Codesign

6.2 Software-Architektur

Die hier erarbeitete Software-Architektur des im Maximalausbau aus Simulationsprozessor und Mikrocontroller bestehenden Zwei-Prozessorsystems ist in Abbildung 6-24 dargestellt. Wesentliche Bestandteile neben der Anwendersoftware sind das Betriebssystem, die Services, die Hardwaretreiber sowie die an der Kommunikation beteiligten Komponenten. In den folgenden Abschnitten wird auf die einzelnen Bestandteile genauer eingegangen.

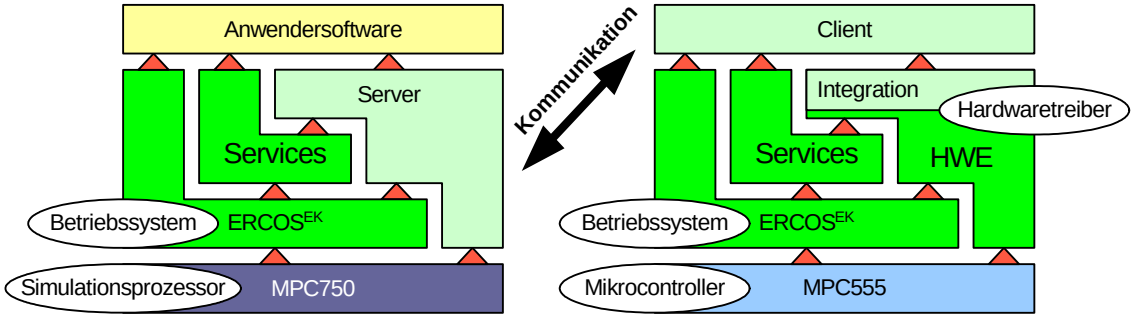


Abbildung 6-24: Softwarestruktur des Entwicklungssystems

6.2.1 Betriebsystem des Entwicklungssystems

Als Betriebssystem wird sowohl für den Mikrocontroller MPC555, als auch für den Simulationsprozessor MPC750 das OSEK konforme Echtzeitbetriebssystem ERCOS^{EK} der Firma ETAS eingesetzt (vergl. [Eta02_1], [Eta02_2], [Eta02_3]). Das Echtzeitbetriebssystem ERCOS^{EK} wird unter anderem verwendet, da es bereits in zahlreichen Serienanwendungen erfolgreich im Einsatz ist und sowohl für den Mikrocontroller MPC555, als auch für den Simulationsprozessor MPC750 zur Verfügung steht.

Durch die Verwendung eines einheitlichen Betriebsystems für Mikrocontroller und Simulationsprozessor wird das Verschieben der Anwendersoftware zwischen den beiden Rechnersystemen vereinfacht.

6.2.2 Hardwaretreiber

Es werden die Hardwaretreiber aus aktuellen Serienprojekten von Diesel- bzw. Benzinmotorsteuergeräten der Robert BOSCH GmbH verwendet. Diese Hardwaretreiber wurden unter dem Namen „Hardware Encapsulation“ (HWE) von den Geschäftsbereichen für Dieselsysteme (DS) und Benzinsysteme (GS) entwickelt. Die Aufgaben der Hardwarekapselung sind:

- Analogwerterfassung,
- Digitalwerterfassung und -ausgabe,
- PWM-Signalwerterfassung und -ausgabe,
- CAN-Kommunikation,
- Bedienung der seriellen Schnittstellen,
- EEPROM-Bearbeitung,
- Flashprogrammierung,
- Mikrocontroller-Selbsttest.

Weitergehende Informationen sind in Anhang 10 zu finden.

Um den Einsatz, der als Library vorliegenden HWE zu erleichtern, wurde diese im Rahmen dieser Arbeit unter Verwendung von Klassen in das ASCET integriert. ASCET ist ein grafisches Entwicklungswerkzeug für die funktionale Entwicklung eingebetteter Softwaresysteme der Firma ETAS (vergl. [Eta02_1], [Eta02_2], [Eta02_3]).

Der Aufbau, die Funktionsweise sowie die Integration der HWE werden im Folgenden am Beispiel der Analogwerterfassung veranschaulicht. Dazu werden exemplarisch einige ADC-Klassen in einem Blockdiagramm instanziiert. In Abbildung 6-25 ist der Blockdiagramm Editor (BDE) der grafischen Entwicklungsumgebung ASCET abgebildet. Im rechten Teil des Fensters sind die grafischen Repräsentationen der Instanzen der ADC-Klassen zu sehen. Es ist zu erkennen, dass jede ADC-Klasse drei Ausgänge besitzt (Status Wert, physikalischer Wert ADC Register). Alle Ausgänge sind mit entsprechenden Variablen verbunden und werden somit der Anwendung zur Verfügung gestellt. Bei einer späteren optimierten Anwendung kann entweder auf den Registerwert oder den physikalischen Wert verzichtet werden. Im linken oberen Teil des Fensters ist eine Liste mit sämtlichen in diesem Blockdiagramm verwendeten Elementen (Klassen, Variable, etc.) abgebildet. Der linke untere Teil des Fensters zeigt die in diesem Blockdiagramm definierten und somit zur Verfügung stehenden Prozesse.

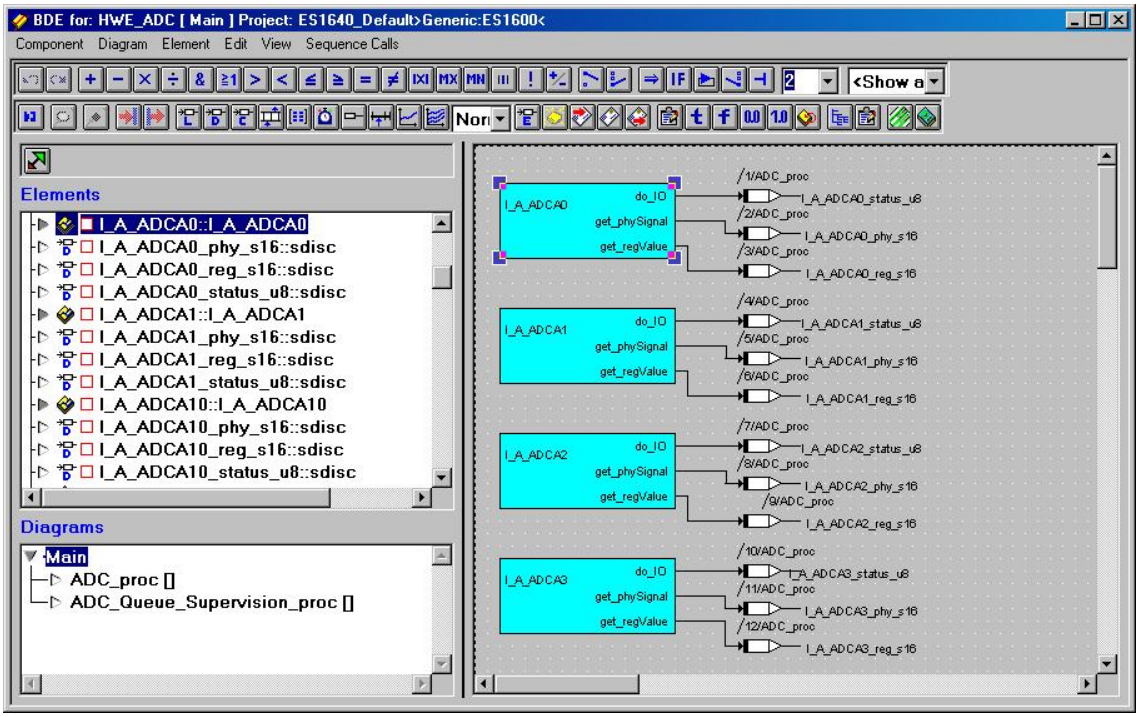


Abbildung 6-25: Blockdiagramm ADC-Klassen

6.2.3 Einsatz der Service Routinen

Die Services sind unterstützende und sehr gut optimierte Routinen unter anderem für arithmetische Berechnungen, zum Filtern von Signalen sowie zur Integration und Interpolation. Die als Library vorliegenden Services wurden im Rahmen dieser Arbeit in das Mikrocontroller Projekt so eingebunden, dass sie in C-Code Modulen und Klassen direkt verwendet werden können. Da diese Library nur für den Mikrocontroller, nicht aber für den Simulationsprozessor verfügbar ist, und zudem die Funktionsentwicklung häufig auf grafischer Ebene erfolgt, werden die am häufigsten benötigten Routinen als Klassen mit identischer Funktionalität zur Verfügung gestellt. Diese Klassen können sowohl bei der grafischen Entwicklung von Mikrocontroller- als auch bei Simulationsprozessorprojekten verwendet werden. Funktionen, die unter Verwendung dieser Routinen entwickelt werden, können relativ einfach zwischen dem Mikrocontroller und dem Simulationsprozessor verschoben werden, da der Codegenerator daraus spezifischen Code für den jeweiligen Prozessor oder Mikrocontroller generieren kann. In Abbildung 6-26 sind exemplarisch einige Services zur grafischen Entwicklung dargestellt. Ein detaillierter Überblick über die verfügbaren Services ist im Anhang zu finden.

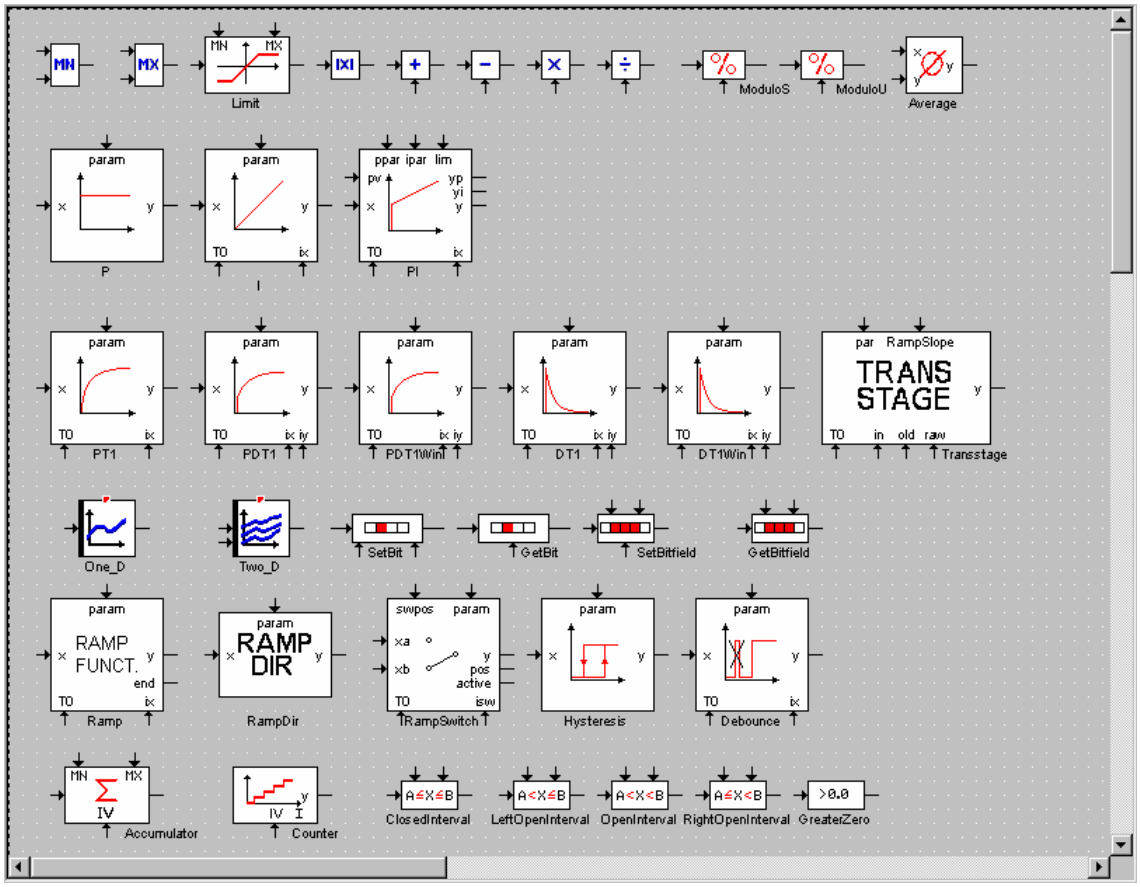


Abbildung 6-26: Services für grafische Entwicklung

Im Folgenden wird anhand eines kleinen regelungstechnischen Beispiels die Anwendung der Services dargestellt. In Abbildung 6-27 ist der Aufbau der Anwendung zu sehen. Ein Rechteckgenerator dient als Sollwertvorgabe. Zur Vereinfachung wird als Regelstrecke ein Verzögerungsglied 2. Ordnung (PT2) der Services verwendet. Die Regelstrecke wird hier simuliert, bei einer realen Anwendung wäre die Regelstrecke natürlich extern.

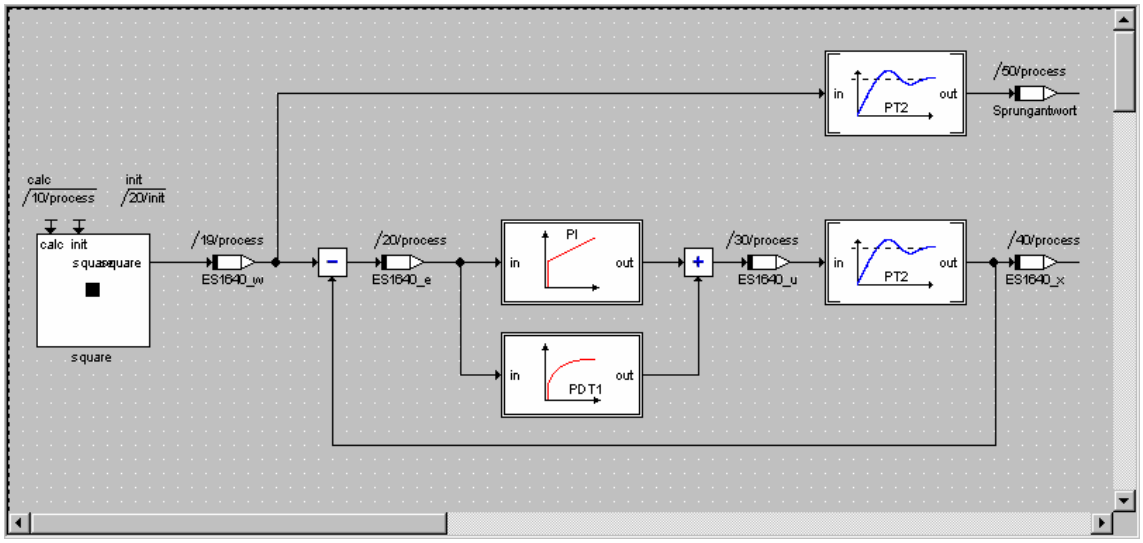


Abbildung 6-27: Beispiel für die Anwendung der Services

Der obere Zweig von Abbildung 6-27 dient zur Aufnahme der Sprungantwort des PT2-Gliedes ohne Regler. Im unteren Zweig von Abbildung 6-27 wird ein PIDT1 Regler aus Basiselementen der Services realisiert. Die Sprungantworten mit und ohne Regler sind in Abbildung 6-28 dargestellt.

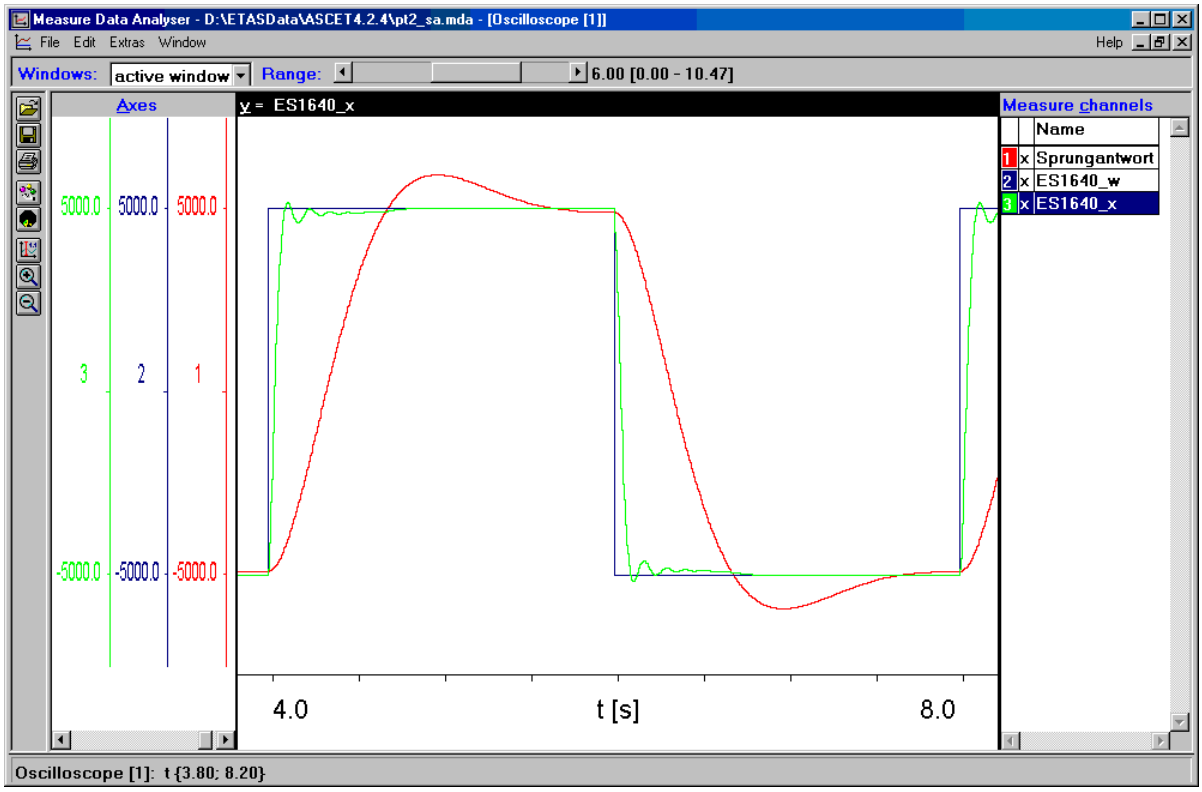


Abbildung 6-28: Sprungantwort

Wie mit diesem Beispiel gezeigt wurde ist es sehr einfach möglich, beispielsweise Regelungen mit Hilfe der auf Laufzeitanforderungen und Speicherplatz optimierten Services zu realisieren.

6.2.4 Kommunikation zwischen Mikrocontroller und Simulationsprozessor

Der nachfolgend dargestellte Kommunikationsmechanismus beschreibt den im Rahmen dieser Arbeit konzipierten und realisierten Datenaustausch zwischen dem Mikrocontroller und dem Simulationsprozessor. Eine wichtige Eigenschaft dieser Kommunikationsschnittstelle ist die einfache und schnelle Auswahl der Kommunikationsgrößen. Das bedeutet unter anderem, dass die benötigten Kommunikationsgrößen ohne Änderungen am Mikrocontrollerprogramm ausgewählt werden können. Das ist erforderlich, wenn in frühen Entwicklungsphasen die Anwendersoftware auf dem Simulationsprozessor und die hardwarenahen Softwarekomponenten auf dem Mikrocontroller ausgeführt werden und Änderungen in der Anwendersoftware vorgenommen werden. In diesem Fall werden die benötigten Eingangs- sowie Ausgangsgrößen auf dem Simulationsprozessor ausgewählt und damit die Kommunikationsschnittstelle konfiguriert. Der Kommunikationstreiber auf dem Mikrocontroller muss daher über den Simulationsprozessor parametriert werden können. Der Datenaustausch für Nutz- und Konfigurationsdaten zwischen dem Mikrocontroller und dem Simulationsprozessor findet über ein von beiden beschreib- und lesbares Dual Port RAM (DPR) statt. Die Kommunikation läuft nach einem an den externen Bypass angelehnten DISTAB-Verfahren ab. Der prinzipielle Ablauf sowie die Speicheraufteilung des DPR sind exemplarisch für ein Raster in Abbildung 6-29 dargestellt.

1. Beim Starten des Bypass-Experiments auf dem Simulationsrechner werden die Pointerliste und die Anzahl der entsprechenden 8 Byte-, 4 Byte-, 2 Byte- und 1 Byte-Signale für die Bypass-Eingangsdaten beschrieben. Die Pointerliste enthält die Adressen der Signale, vom ersten 8 Byte-Signal bis zum letzten 1 Byte-Signal. Jede Adresse ist 4 Byte breit.
2. In diesem Schritt werden die Offset- und Rasterinformationen für die Bypass-Ausgangsdaten beschrieben.
3. Anschließend wird das Aktiv-Flag gesetzt. Dadurch wird die Kommunikation zwischen Mikrocontroller und Simulationsrechner in Gang gesetzt. Schritt 1, 2 und 3 werden nur einmal, zu Beginn eines Bypass-Experiments, ausgeführt.
4. Der Mikrocontroller fragt das Flag-Byte der Pointerliste kontinuierlich ab. Sobald das Aktiv-Flag gesetzt ist, wird der Bypass aktiviert und die Übertragung der Daten beginnt.
5. Die über die Pointerliste vorgegebenen Bypass-Eingangsdaten werden vom Mikrocontroller in den dafür vorgesehenen Datenpuffer (CHNL_EB) kopiert. Die Daten werden nacheinander in der Reihenfolge geschrieben, die in der Liste vorgegeben ist. Zuerst werden die 8 Byte-Signale geschrieben, gefolgt von den 4 Byte-, 2 Byte- und 1 Byte-Signalen.
6. Die Trigger ID wird vom Mikrocontroller auf die entsprechende Adresse des TRIG Segments geschrieben. Es werden bis zu 16 Trigger IDs unterstützt. Trigger ID1 wird mit eins beschrieben, Trigger ID2 mit zwei und so weiter. Eine Null bedeutet, dass diese Trigger ID derzeit nicht aktiv ist.
7. Der Trigger wird vom Mikrocontroller auf die entsprechende Adresse des TRIG Segments geschrieben. Analog zur Trigger ID werden bis zu 16 Trigger unterstützt. Trigger1 wird ebenfalls mit eins beschrieben, Trigger2 mit zwei und so weiter. Eine Null bedeutet, dass dieser Trigger derzeit nicht aktiv ist.
8. Anschließend wird beim Simulationsrechner ein Interrupt ausgelöst.
9. Durch Auslesen der Trigger ID stellt die Interrupt Service Routine des Simulationsrechners fest, welcher Kanal angestoßen wurde, setzt die Trigger ID zurück und aktiviert die zugehörige Software Task. Innerhalb dieser Task wird dann der entsprechende Datenpuffer ausgelesen und ebenfalls der Trigger zurückgesetzt. Mit diesem Mechanismus kann der Mikrocontroller erkennen, ob der Interrupt bearbeitet und die Daten abgeholt wurden. Anhand der Informationen aus der ASAM-MCD-2MC-Datei (z.B. der Umrechnungsformel) kann der Simulationsrechner die Rohdaten aus dem Mikrocontroller in physikalische Modellgrößen umrechnen.

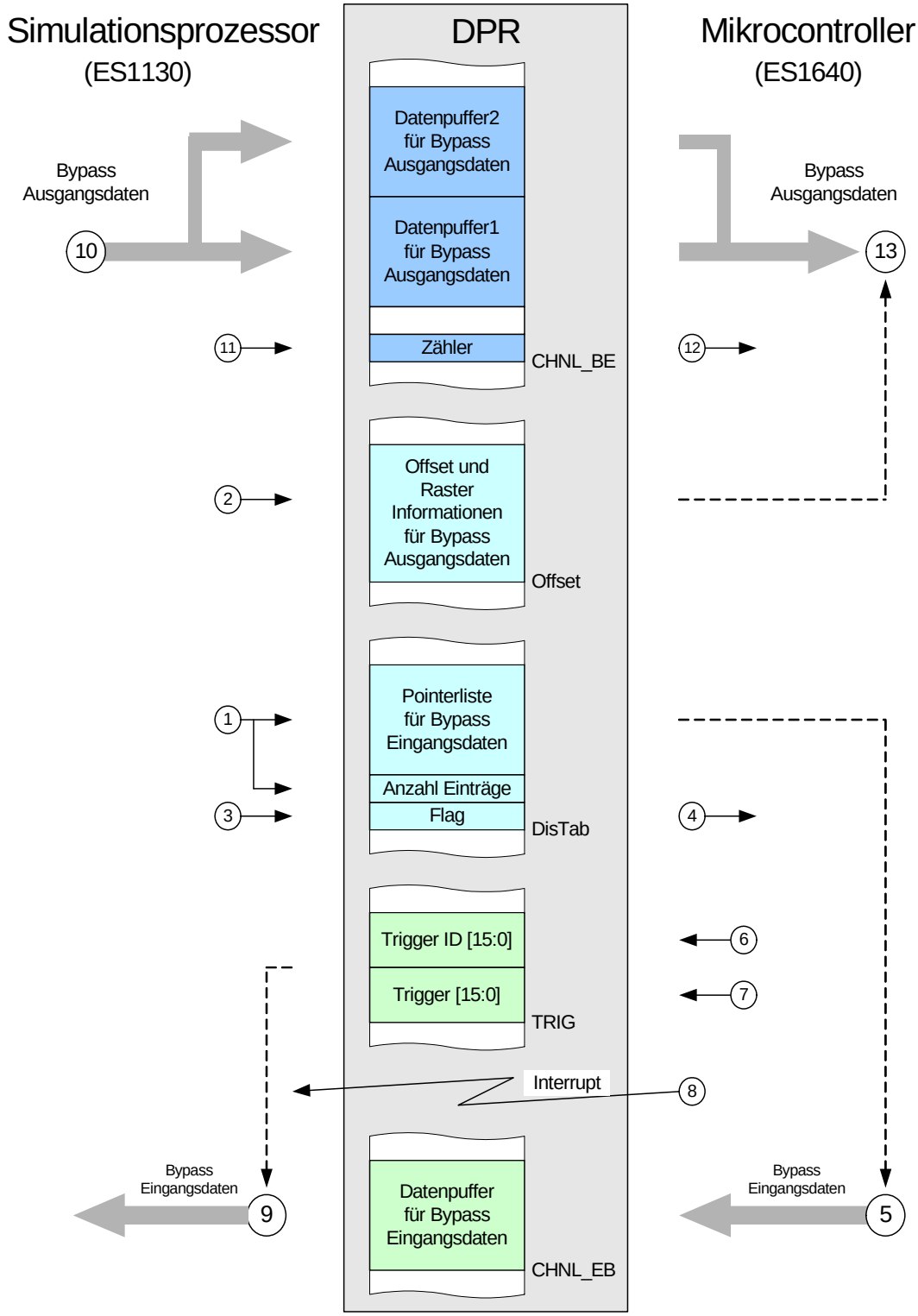


Abbildung 6-29: Kommunikationsablauf

10. Der Simulationsrechner berechnet die ins Bypass-Modell ausgelagerten Funktionen und schreibt anschließend die Ergebnisse als Bypass-Ausgangsdaten in einen der beiden Datenpuffer. Für jeden Bypass-Kanal gibt es zwei Datenpuffer, die abhängig vom Zählerstand des Kommunikationszählers beschrieben werden. Ist der Zähler ungerade, wird der untere Datenpuffer beschrieben, ist er gerade, der Obere. Für die Bypass-Ausgangsdaten werden zwei Datenpuffer benötigt, da dieser Vorgang unsynchronisiert erfolgt. Die Ergebnisse der Berechnungen im Simulationsrechner werden zurückgeschrieben wann immer diese Berechnungen abgeschlossen sind. Dadurch kann es vorkommen, dass die Ergebnisse vom Mikrocontroller abgefragt werden bevor oder während die Daten zurückgeschrieben werden. Deshalb wird

immer eine Kopie der Daten in einem Datenpuffer gehalten, bis das Zurückschreiben der neuen Daten beendet ist, um die Datenkonsistenz der Bypass-Ausgangsdaten sicherzustellen.

11. Nachdem sämtliche Daten in den Datenpuffer kopiert wurden, wird der Zählerstand in das DPR kopiert.
12. Der Wert des Zählers wird vom Mikrocontroller kontinuierlich ausgelesen. Aufgrund des Werts(gerade/ungerade) entscheidet der Mikrocontroller, ob und in welchem Datenpuffer neue Daten vorliegen.
13. Die Bypass-Ausgangsdaten werden vom Mikrocontroller aus dem entsprechenden Datenpuffer ausgelesen und auf die durch die Offsets festgelegten Werte kopiert.

Der zeitliche Ablauf sowie die bei der Kommunikation auftretenden Verzögerungszeiten sind in Abbildung 6-30 dargestellt.

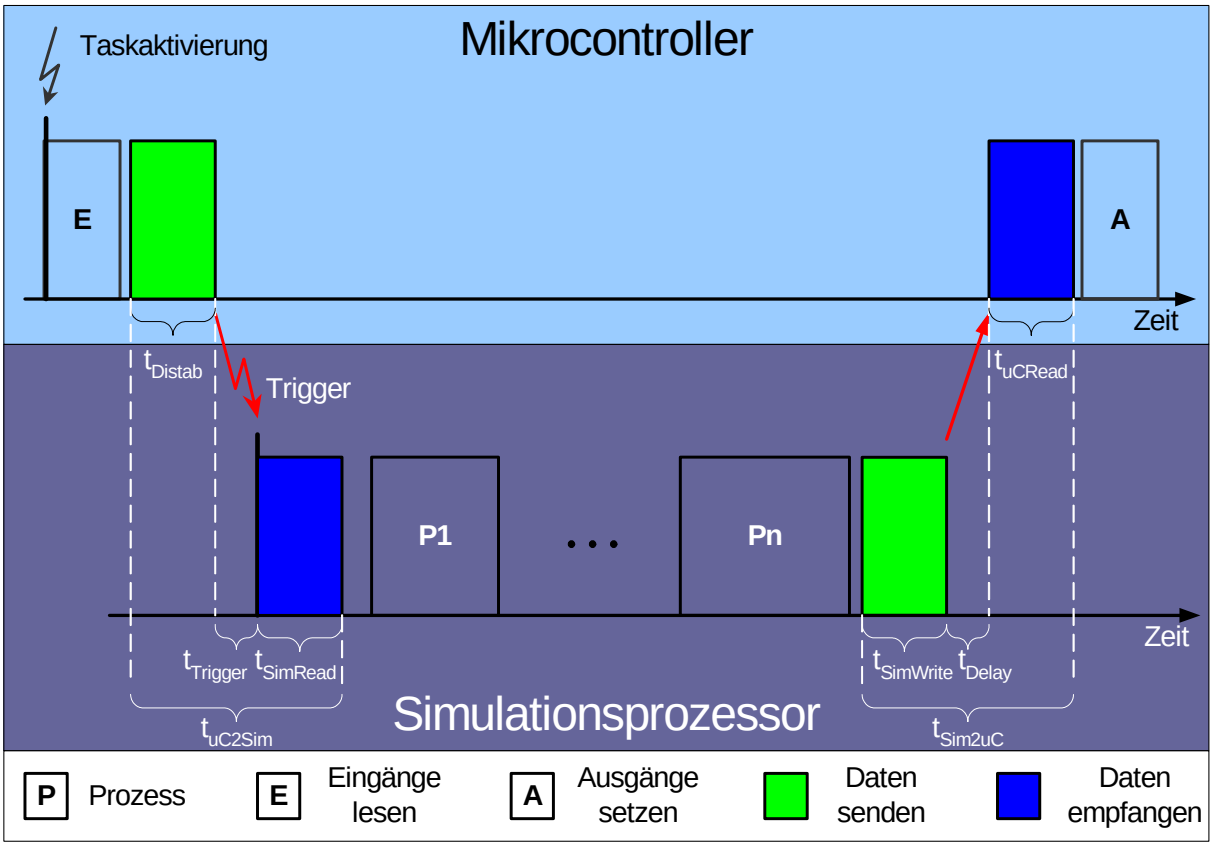


Abbildung 6-30: Kommunikationszeiten

t_{Distab}: Zeit, welche der Mikrocontroller benötigt, um die in der Display-Tabelle festgelegten Daten in den Datenpuffer zu kopieren. Diese Zeit setzt sich aus einem Offset, um die Display-Tabelle zu analysieren, und einem Anteil, welcher proportional zur Datenmenge ist, zusammen. (Abbildung 6-31)

$t_{Trigger}$: Verzögerungszeit, bis der Simulationsprozessor auf den Interrupt des Mikrocontrollers reagiert. Die Verzögerungszeit setzt sich aus der Interrupt Latenzzeit des Mikrocontrollers und des Echtzeitbetriebssystems zusammen und ist daher unabhängig von der Datenmenge. (Abbildung 6-32)

$t_{SimRead}$: Zeit, um Daten aus dem DPR Puffer des Mikrocontrollers zu lesen und in entsprechenden Variablen auf dem Simulationsprozessor zur Verfügung zu

stellen. Diese Zeit besteht ebenfalls aus einem Offset und einem zur Datenmenge proportionalen Anteil. (Abbildung 6-33)

$t_{SimWrite}$: Zeit, um Daten aus Variablen des Simulationsprozessors in den Puffer des Mikrocontrollers zu schreiben. Auch diese Zeit besteht aus einem Offset und einem zur Datenmenge proportionalen Anteil. (Abbildung 6-34)

t_{Delay}: Verzögerungszeit, bis der Mikrocontroller feststellt, dass neue Daten vom Simulationsprozessor vorliegen. Die Verzögerungszeit ist prinzipbedingt unabhängig von der Datenmenge, da der Mikrocontroller hier in einer Schleife ein Flag abfragt, bis dieses vom Simulationsprozessor aktiviert wird. Diese Zeit kann daher auch als konstant betrachtet werden. (Abbildung 6-35)

t_{uCRead}: Zeit, um Daten aus dem DPR Puffer zu lesen und in entsprechenden Variablen auf dem Mikrocontroller zur Verfügung zu stellen. Wie die restlichen Lese- und Schreibzeiten, besteht diese Zeit aus einem Offset und einem zur Datenmenge proportionalen Anteil. (Abbildung 6-36)

t_{uC2Sim}: Summe aller Zeiten, um Daten vom Mikrocontroller zum Simulationsprozessor zu übertragen. (Abbildung 6-37)

t_{Sim2uC}: Summe aller Zeiten, um Daten vom Simulationsprozessor zum Mikrocontroller zu übertragen. (Abbildung 6-38)

In den folgenden Abbildungen sind Mittelwerte aus jeweils drei Messungen für die einzelnen Zeiten und Datenmengen dargestellt. Die blauen Punkte sind die Mittelwerte der real gemessenen Zeiten und die roten Linien sind durch lineare Gleichungen approximierte Kennlinien.

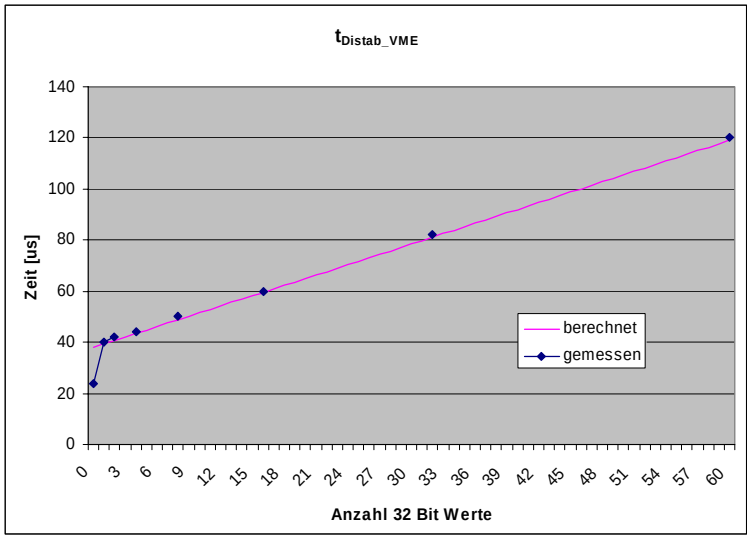


Abbildung 6-31: t_{Distab}

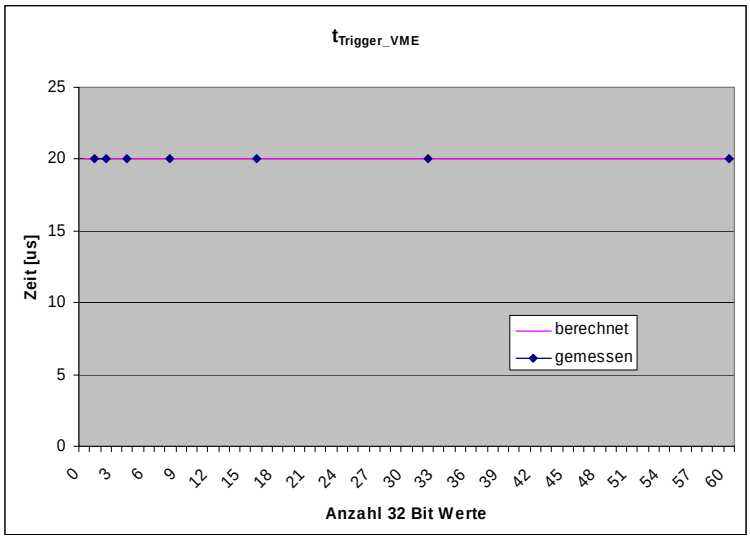


Abbildung 6-32: $t_{Trigger}$

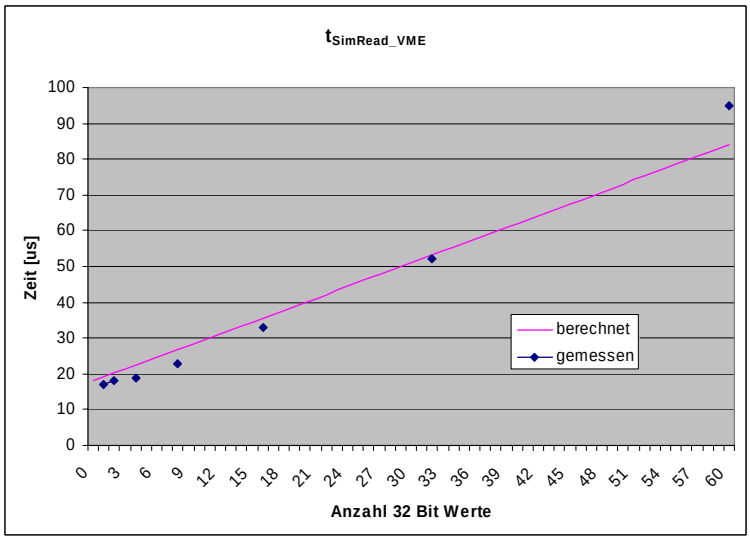


Abbildung 6-33: $t_{SimRead}$

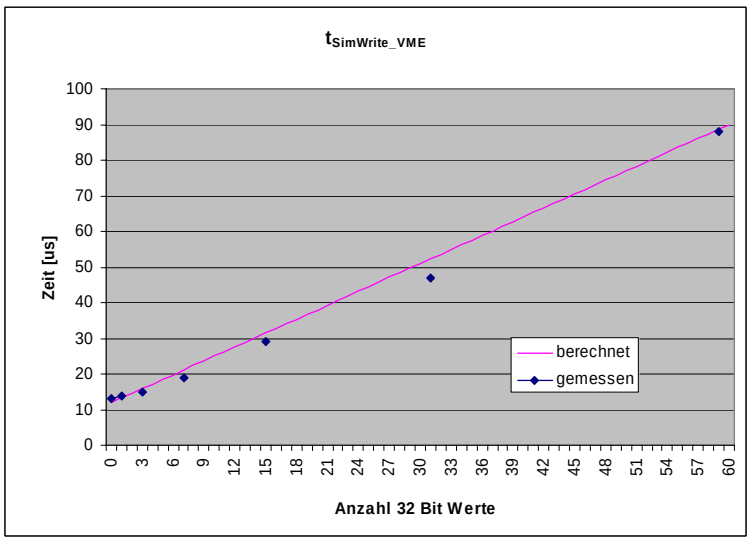


Abbildung 6-34: $t_{SimWrite}$

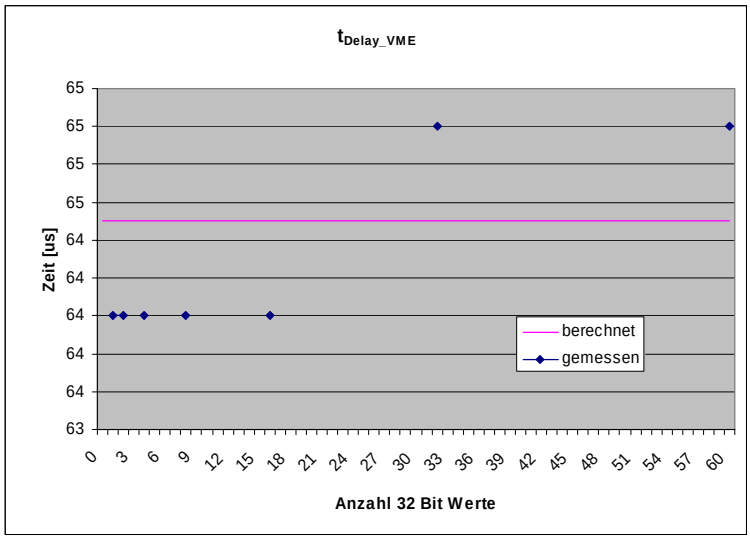


Abbildung 6-35: t_{Delay}

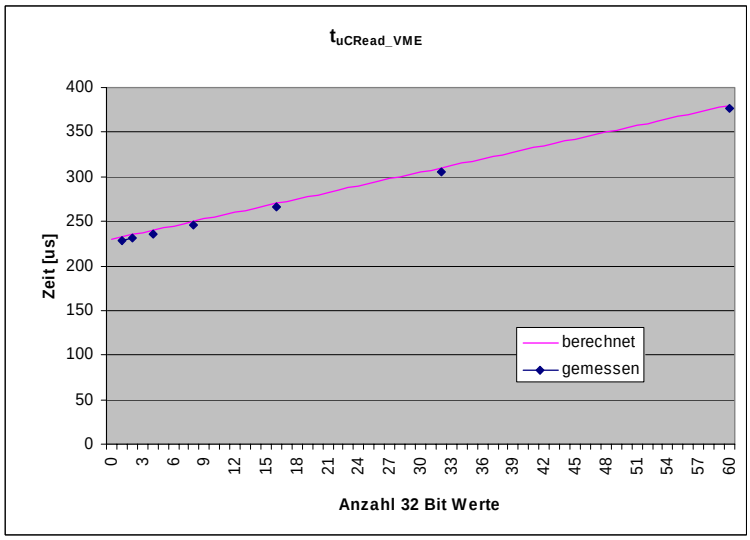


Abbildung 6-36: t_{uCRead}

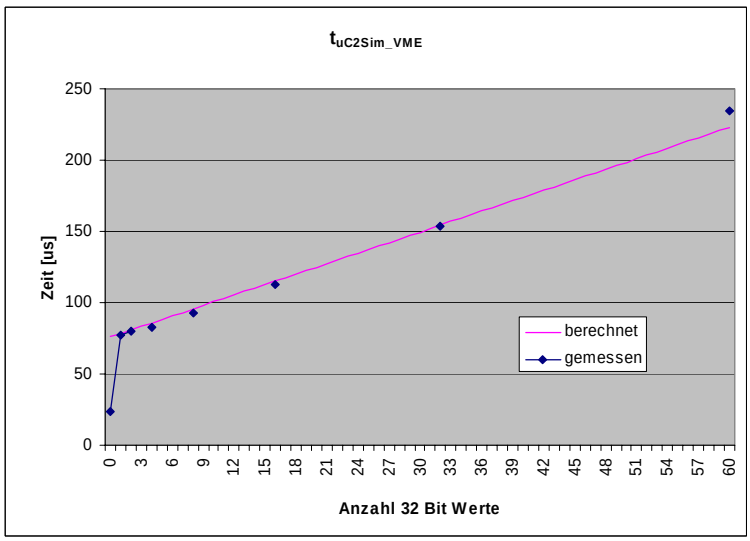


Abbildung 6-37: t_{uC2Sim}

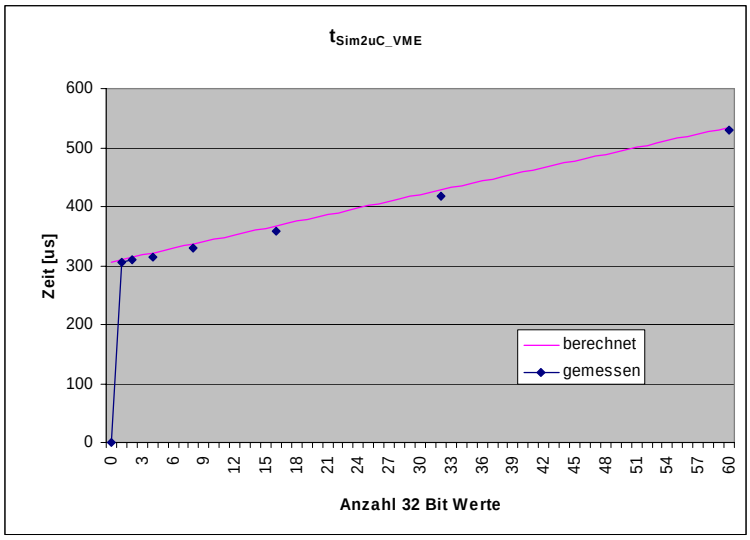


Abbildung 6-38: t_{Sim2uC}

Es lässt sich in guter Näherung eine lineare Abhängigkeit der benötigten Kommunikationszeiten von der übertragenen Datenmenge nachweisen. Nachfolgend werden die erzielten Übertragungszeiten denen des heute standardmäßig verwendeten ETK Bypass gegenübergestellt.

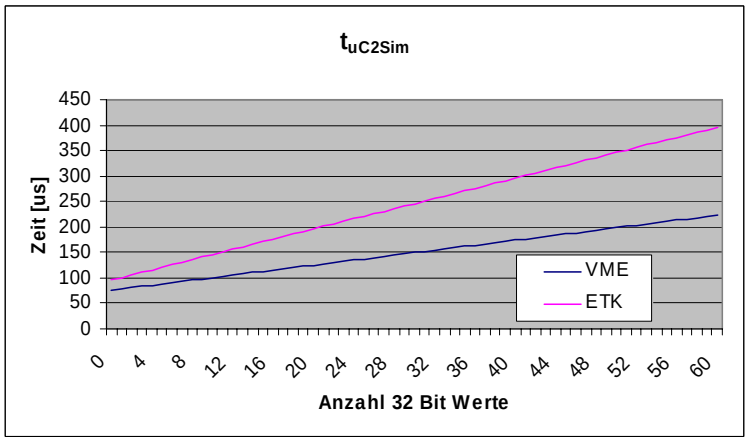


Abbildung 6-39: t_{uC2Sim}

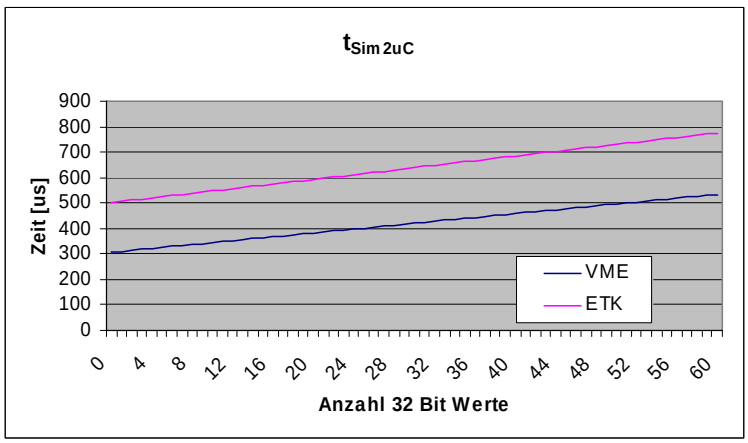


Abbildung 6-40: t_{Sim2uC}

Man kann erkennen, dass sowohl die Übertragung vom Mikrocontroller zum Simulationsprozessor, als auch in der entgegen gesetzten Richtung deutlich schneller ist, als beim bereits sehr schnellen 100 Mbit ETK Bypass.

Das Abholen der Bypasswerte (Freischnitt) sowie die Umschaltung zwischen den Bypasswerten und den Ersatzwerten wird in C-Klassen realisiert. Es wurden für alle unterstützten Datentypen entsprechende Klassen vorbereitet. In Abbildung 6-41 sind in einem ASCET Modul exemplarisch einige Bypass Klassen für unterschiedliche Datentypen dargestellt.

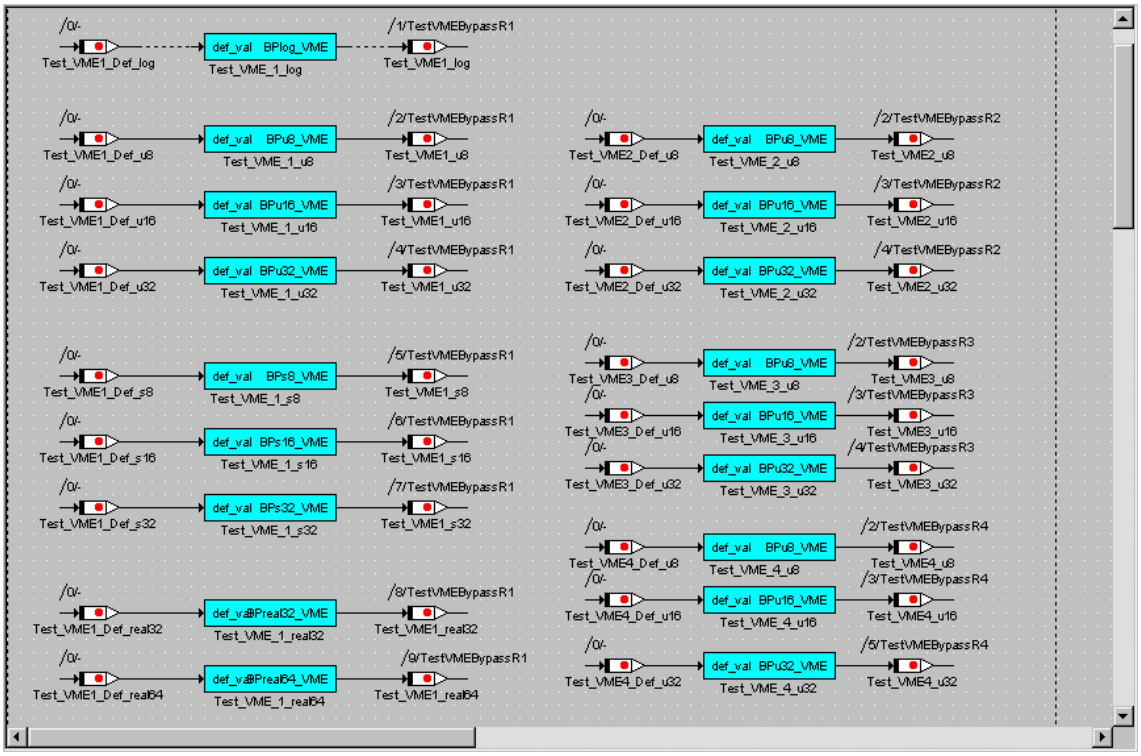


Abbildung 6-41: ASCET Modul mit Bypass Klassen

Durch Einhalten spezieller Namensregeln für die Kommunikationsklassen kann die Anpassung der für die Spezifikation der Kommunikation verwendete ASAM-MCD-2MC automatisiert werden. Die Namensregeln sind in Abbildung 6-42. dargestellt.

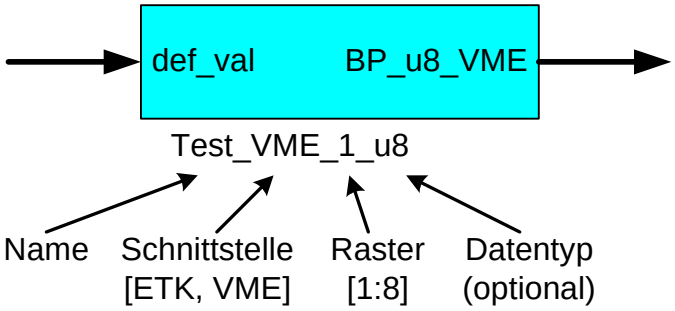


Abbildung 6-42: Namensregeln für Bypass Klassen

Der C-Code für eine Bypass Klasse ist in Abbildung 6-43 abgebildet. In dieser Klasse wird die Umschaltung zwischen dem Bypasswert und dem Ersatzwert realisiert.

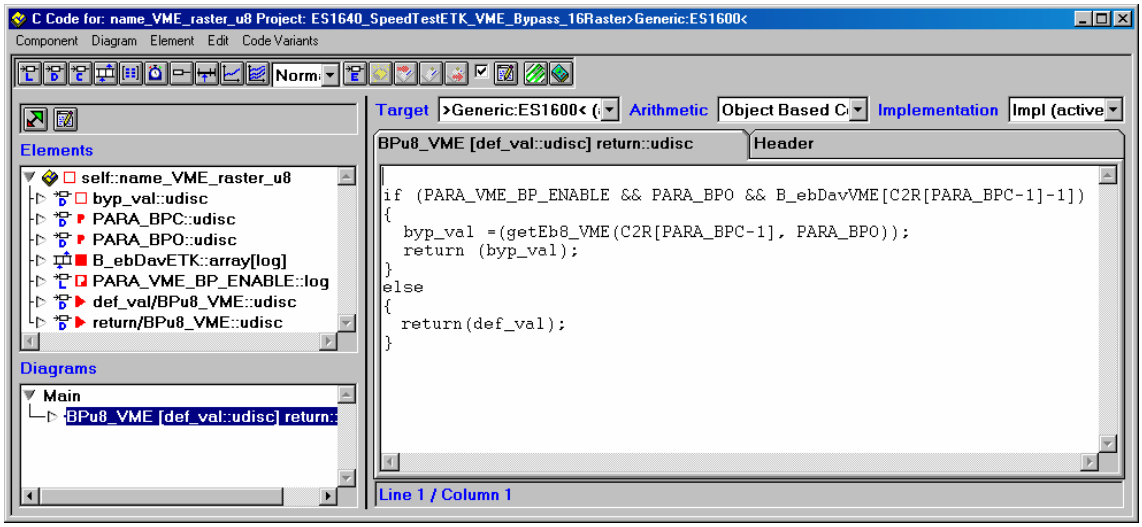


Abbildung 6-43: ASCET C-Code der Bypass Klasse

Das Abholen der Bypasswerte aus dem DPR wird in einer externen C Funktion (getEb8_VME) vorgenommen. Der C-Code dieser Funktion ist in Abbildung 6-44 zu sehen.

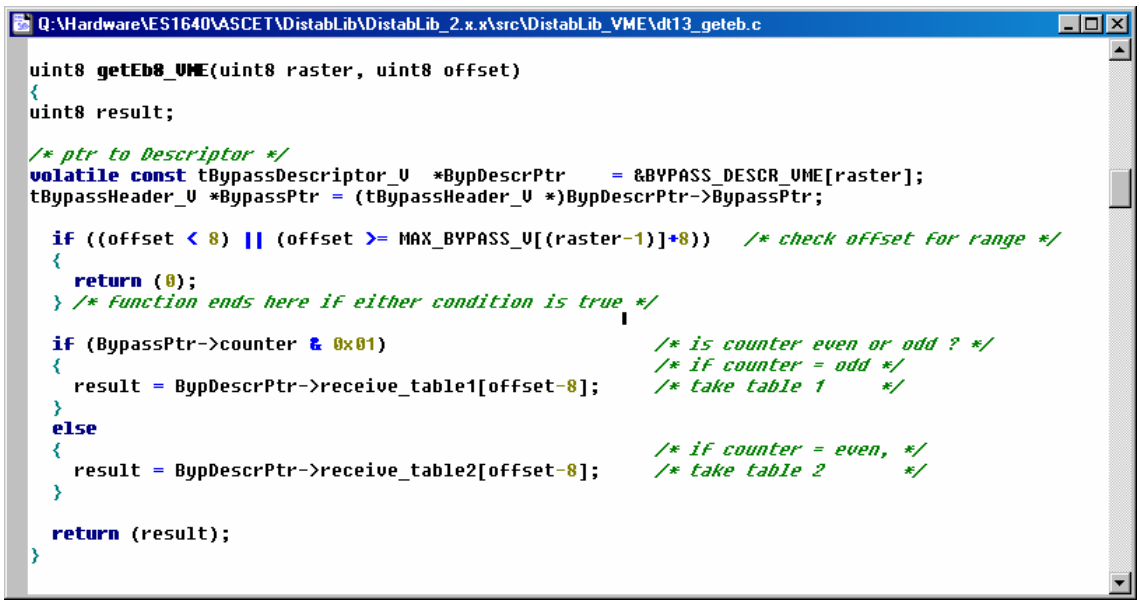


Abbildung 6-44: Externer C-Code der Bypass Klasse

6.2.5 Sicherheitskonzept

Um zu vermeiden, dass ungültige oder unplausible Werte vom Simulationsprozessor auf dem Mikrocontroller verwendet werden, wurde das nachfolgend beschriebene Sicherheitskonzept implementiert (siehe Abbildung 6-45). Damit Werte vom Simulationsprozessor (Bypass_Value) verwendet werden, müssen alle der nachfolgend genannten Bedingungen erfüllt sein:

- Der Bypass muss aktiviert sein,
- die entsprechende Variable muss auf dem Simulationsprozessor als Bypassgröße ausgewählt sein,
- es darf nur eine konfigurierbare maximale Anzahl von Fehlern aufgetreten sein.

Ist nur eine Bedingung nicht erfüllt, wird ein Ersatzwert (Default_Value) verwendet. Der Ersatzwert kann eine Konstante sein oder aber vom Mikrocontroller berechnet werden. Bevor der Wert auf dem Mikrocontroller verwendet wird, kann er noch optional auf einen für die Anwendung gültigen Wertebereich begrenzt werden.

Für sicherheitskritische Anwendungen kann beim Auftreten eines Fehlers auch die komplette Anwendung neu gestartet oder z.B. durch deaktivieren kritischer Ausgänge in einen sicheren Zustand gebracht und angehalten werden.

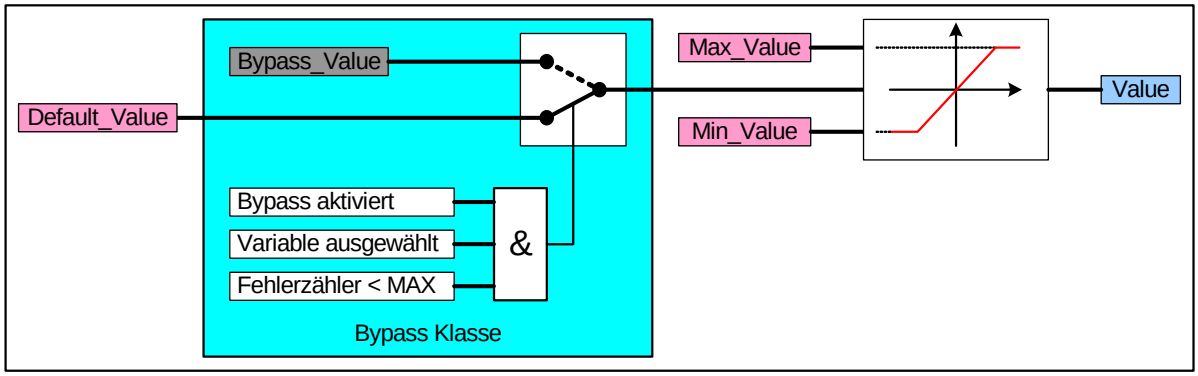


Abbildung 6-45: Sicherheitskonzept

6.2.6 Anwendersoftware

Sowohl die Software für den Mikrocontroller, als auch die Software für den Simulationsprozessor, wird mit dem grafischen Entwicklungswerkzeug ASCET der Firma ETAS erstellt. Durch die Verwendung einer einheitlichen Entwicklungsumgebung wird das Verschieben einzelner Funktionen der Anwendersoftware zwischen Mikrocontroller und Simulationsprozessor erheblich erleichtert. Es wird die jeweils für eine entsprechende Aufgaben am besten passende Beschreibungsform verwendet (Blockdiagramm, Zustandsautomat, Textuelle Programmiersprache (z.B. „C“))

6.2.7 Transferieren von Anwendersoftware

In diesem Abschnitt wird anhand eines einfachen regelungstechnischen Beispiels gezeigt, wie die Softwareentwicklung schnell und komfortabel auf dem Simulationsprozessor durchgeführt und anschließend auf den Mikrocontroller übertragen werden kann. Um das Beispiel einfach zu halten, wird die Sollwertvorgabe als Rechteckgenerator und die Regelstrecke als PT2 Glied in Software auf der ES1640 realisiert. Als Regler wird ein PID Regler verwendet, welcher über eine Freischnittklasse auf den Simulationsprozessor ausgelagert ist. Das Blockdiagramm dieses Aufbaus ist in Abbildung 6-46 zu sehen.

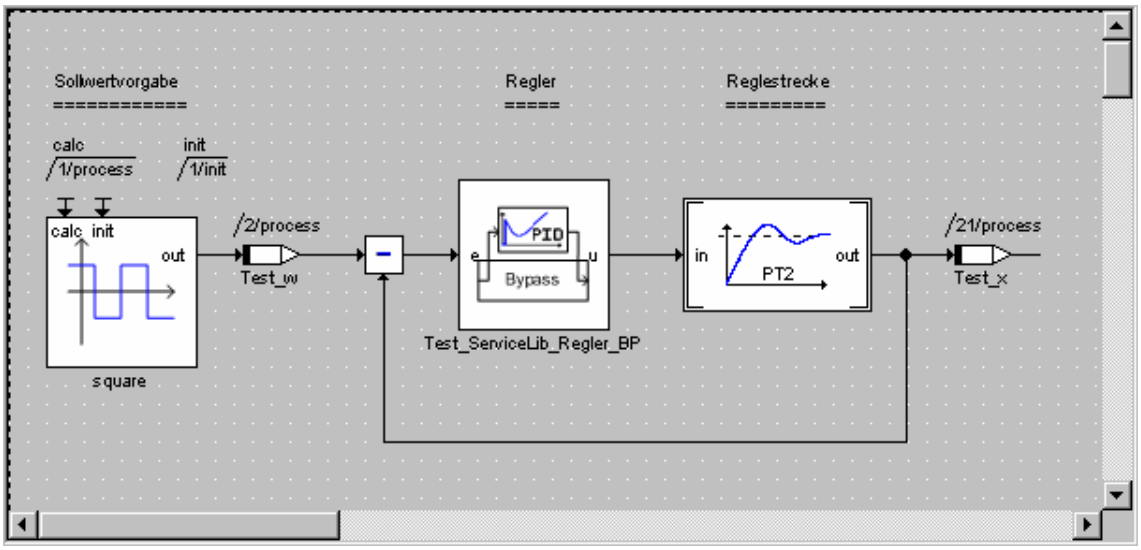


Abbildung 6-46: ES1640 Anwendersoftware mit Freischnittklasse

Der interne Aufbau der Freischnittklasse des PID Reglers ist in Abbildung 6-47 dargestellt. Weitere Informationen zu der Funktionsweise des Freischnittes sind in 6.2.4 Kommunikation zu finden.

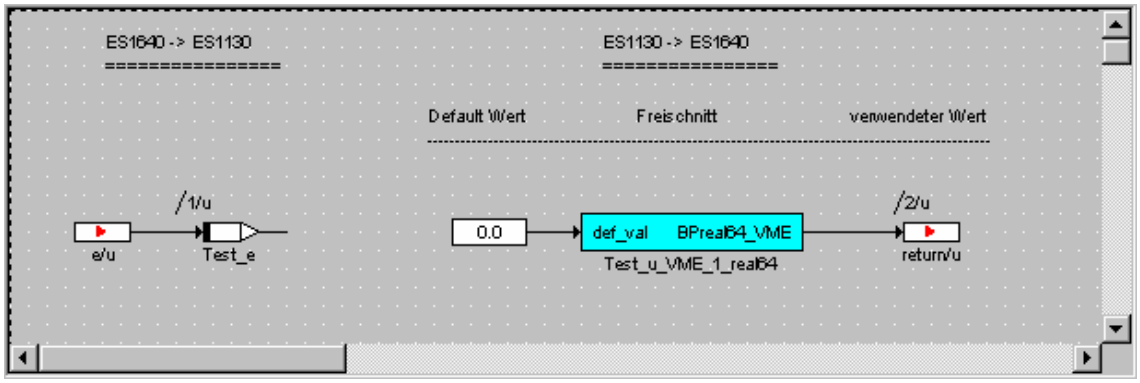


Abbildung 6-47: Freischnittklasse

Auf dem Simulationsprozessor kann der benötigte Wert von dem Mikrocontroller über eine Message (Test_e) abgeholt und dem Regler zugeführt werden. Der vom Regler des Simulationsprozessors berechnete Ausgangswert wird ebenfalls über eine Message (Test_u_VME_1_real64) an den Mikrocontroller übergeben. Auf dem Simulationsprozessor kann sehr schnell und komfortabel ein entsprechender Regler strukturiert und parametriert werden. Die Ein-/ Ausgangsmessages sowie der Regler sind in Abbildung 6-48 zu sehen.

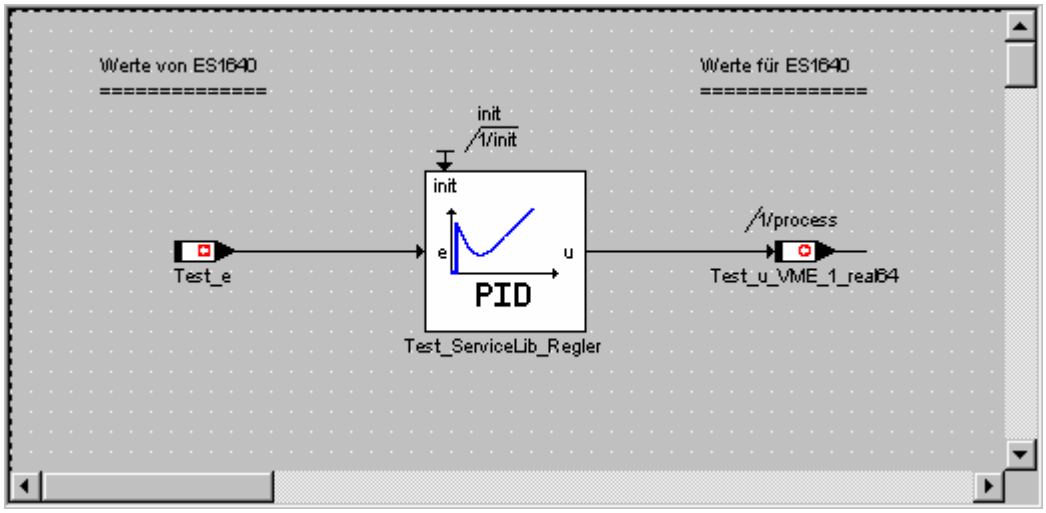


Abbildung 6-48: Regler auf dem Simulationsprozessor

Nachdem ein entsprechender Regler strukturiert, parametriert und verifiziert wurde, kann er auf der ES1640 zum Einsatz kommen. Die in Abbildung 6-46 dargestellte Freischnittklasse wird durch den Regler aus Abbildung 6-48 ersetzt. Daraus ergibt sich dann der in Abbildung 6-49 zu sehende Aufbau.

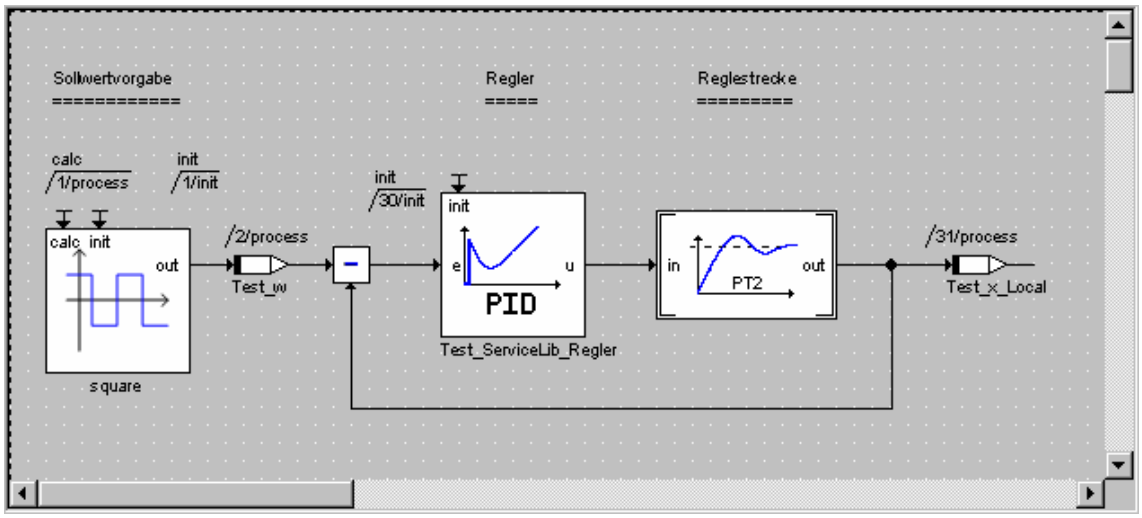


Abbildung 6-49: ES1640 Anwendersoftware mit Regler ohne Freischnitt

In Abbildung 6-50 sind die Sollwertvorgabe (rot) sowie die Sprungantworten des Reglers auf dem Simulationsprozessor (grün) im Bypassbetrieb und dem direkt auf dem Mikrocontroller (gelb) implementierten Regler einander gegenübergestellt. Es wurden exakt dieselben Reglerparameter für den Regler auf dem Simulationsprozessor, als auch für den Regler auf dem Mikrocontroller verwendet. Man kann erkennen, dass selbst ohne Parameteranpassungen das Verhalten der beiden Regler nahezu identisch ist.

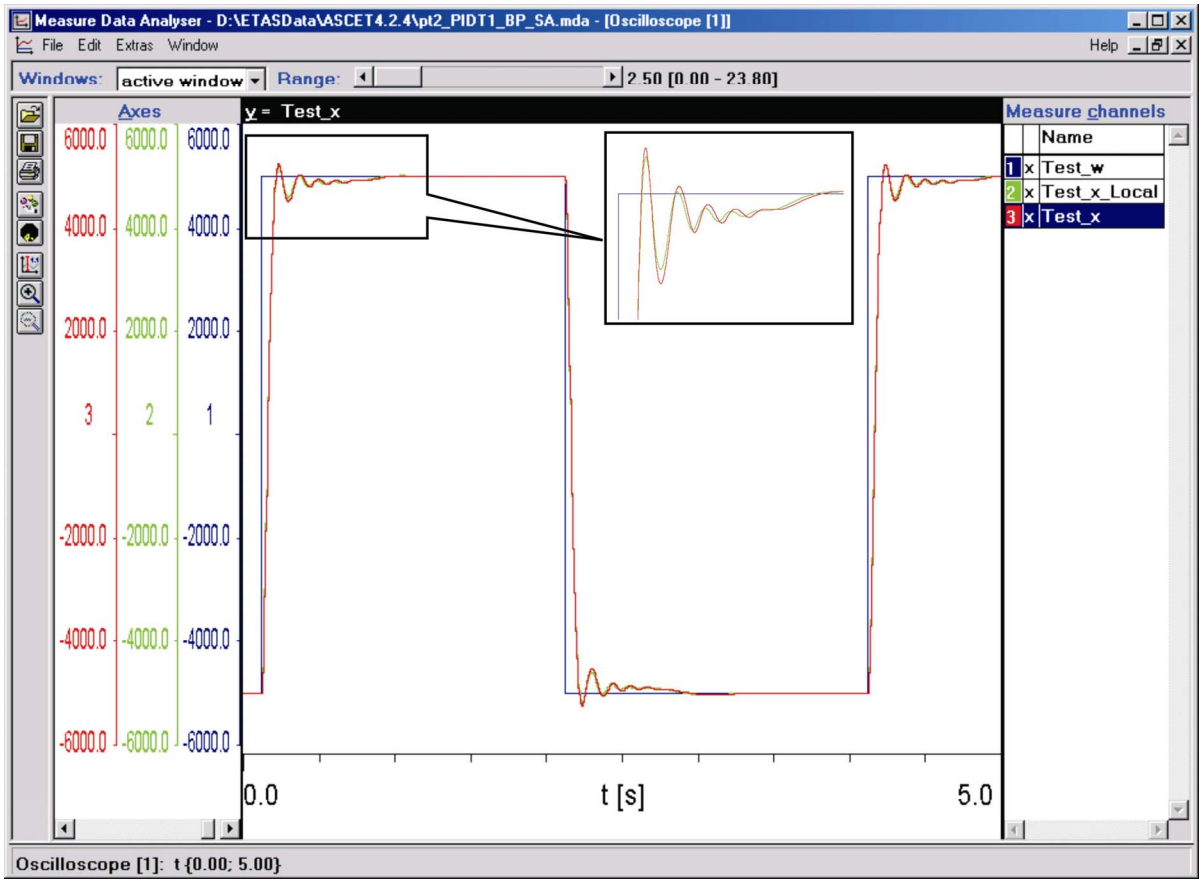


Abbildung 6-50: Sprungantworten

Das Arbeiten auf dem Simulationstarget ist sowohl bzgl. der Programmlaufzeit, als auch bzgl. der Turnaroundzeit (Codegenerierung, Kompilierung, Download) deutlich schneller, als auf dem Mikrocontroller-Target ES1640. Der Geschwindigkeitsvorteil bei der Programmlaufzeit wird durch die deutlich höhere Rechenleistung des Simulationsprozessors ES1135 (2320 MIPS [IBM03]) gegenüber dem Mikrocontroller (52,7 MIPS [Mot00]) erreicht. Die schnellere Turnaroundzeit kann durch folgende Punkte realisiert werden:

- **Codegenerierung deutlich schneller**
  (da durch die höhere Rechenleistung und den größeren Speicher des Simulationsprozessors nicht so viele Optimierungsschritte bei der Codegenerierung durchgeführt werden müssen),
- **integrierte Software Downloadschnittstelle**
  (damit kann die entwickelte Software direkt mit der Entwicklungsumgebung auf den Simulationsprozessor geladen werden und muss nicht mit einem externen Werkzeug wie beispielsweise einem Debugger manuell geladen werden),
- **integrierte Visualisierungs- und Verstellschnittstelle**
  (damit können beliebige Größen im Modell dargestellt und verändert werden).

6.3 Anwendungsbeispiele

6.3.1 Drehzahl und Positionserfassung eines Verbrennungsmotors

Einige der wichtigsten Eingangsgrößen für die Regelung eines Viertakt-Verbrennungsmotors sind dessen Drehzahl sowie die aktuelle Position der Kolben. Diese beiden Größen werden üblicherweise mithilfe von Sensoren an der Kurbel- und Nockenwelle gewonnen. Exemplarisch sind die Signalverläufe bei einer konstanten Drehzahl in Abbildung 6-51 dargestellt.

Im rechten Bild sind 2 volle Umdrehungen der Kurbelwelle zu sehen. Im linken Bild ist der Bereich um die Lücke vergrößert dargestellt. Den Kanälen am Oszilloskop sind folgende Signale zugeordnet:

- Sensorsignal des induktiven Kurbelwellensensors (1)
  (und einem Inkrementgeberrad mit 60-2x2 Zähnen),
- Digitalisiertes Signal des Kurbelwellensensors (4),
- Sensorsignal eines Nockenwellensensors (2).

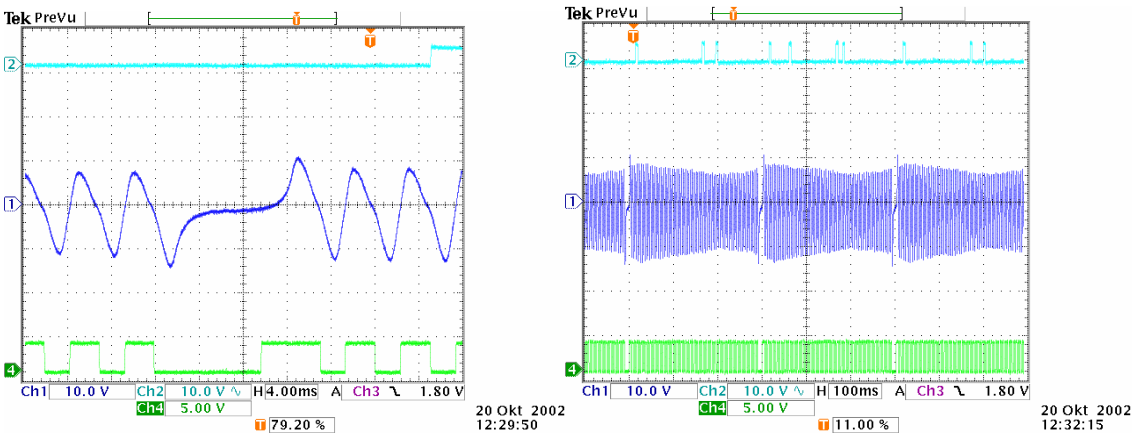


Abbildung 6-51: Kurbel- und Nockenwellensignal bei 200 1/min

Weitere Informationen zu unterschiedlichen Geberrädern und Sensoren sind im Anhang 10.3 zusammengestellt.

6.3.1.1 Auswertung von Kurbel- und Nockenwellensignalen (Winkeluhr)

Mit der Einführung elektronischer Steuerungen für Verbrennungsmotoren wurden Winkeluhren verwendet, um die Position der Kolben darzustellen und alle winkelsynchronen Aktionen, wie beispielsweise die Einspritzung zu kontrollieren. Anfangs wurden die Winkeluhren häufig in Software mit Hilfe von Interruptus realisiert. Die Mikrocontroller waren dadurch häufig schon stark ausgelastet. Als immer mehr und neue Aufgaben durch die Mikrocontroller zu bewältigen waren, wurde ein Teil der Winkeluhrfunktionen in hardwareunterstützte Einheiten der Prozessoren ausgelagert. So wird beispielsweise bei den aktuellen Motorola® Mikrocontrollern die „Timing Processing Unit“ (TPU) zur Realisierung der Winkeluhr verwendet. Diese TPU ist sehr leistungsfähig, jedoch sehr aufwändig zu programmieren. Für den Serieneinsatz stellt das kein so großes Problem dar, da hier von Spezialisten eine Version erstellt werden kann, an der nur wenige Änderungen vorgenommen werden müssen. Für ein Entwicklungssystem ist das aber zu unflexibel. Hier muss es möglich sein, in sehr kurzer Zeit neue Ideen evaluieren zu können. Aus diesem Grund wurde die Winkeluhr bei diesem Ansatz im „Field Programable Gate Array“ (FPGA) der rekonfigurierbaren Logik realisiert.

Aus den aufbereiteten und als TTL-Signale vorliegenden Eingängen der Sensoren von Kurbelwelle (Inkrementgeber) und Nockenwelle (Phasengeber) werden in der Winkeluhr (Abbildung 6-52) folgende Ausgänge generiert:

- Ein 12 Bit Zähler dessen Wert dem **Winkel** der Kurbelwelle entspricht
  (Der Winkelzähler dient als Basis für die Ansteuerung der Einspritzventile sowie zur winkelsynchronen Generierung von Interrupts am Mikrocontroller.),

  Ein Bit entspricht dabei $\frac{720°}{2^{12}} = 0{,}176°$

- Die **Zahnzeit** T_Z von Zahn zu Zahn der Kurbelwelle
  (Mit der Vorgabe, dass das Geberrad in 60 äquidistante Zähne (Z) eingeteilt ist, kann aus dieser Zeit die Drehzahl (n) errechnet werden),

$$n\left[\frac{1}{\text{min}}\right] = \frac{60\left[\frac{\text{s}}{\text{min}}\right]}{60[\text{Z}] * T_Z\left[\frac{\text{s}}{\text{Z}}\right]}$$

- Das **Synchron**-Signal
  (Es wird aktiviert, sobald sich die Winkeluhr auf die Eingangssignale synchronisiert, bzw. deaktiviert hat falls die Winkeluhr die Synchronisation mit den Eingangssignalen verliert. Mit dem Synchron-Signal kann beispielsweise die Einspritzung aktiviert werden).

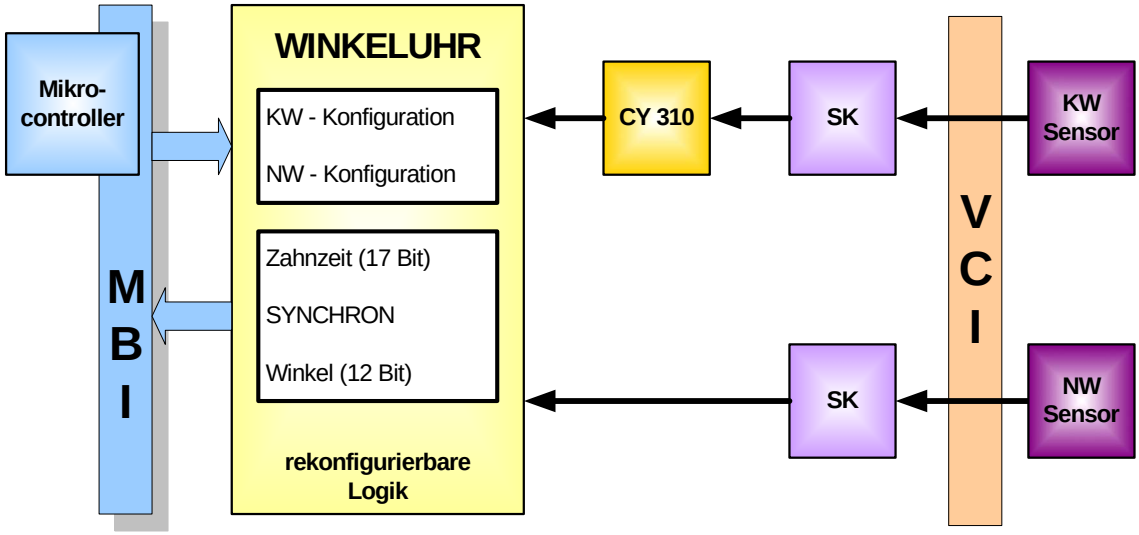


Abbildung 6-52: Winkeluhr

Die Besonderheiten bei der Erfassung von Drehzahl und Position aus Signalen von Seriengeberrädern und Seriensensoren sind:

- Der Inkrementgeber liefert, bedingt durch die Zahnteilung seines Geberrades, alle 6° eine fallende Flanke. Die Auflösung der Winkeluhr soll aber wesentlich feiner (0,176°) sein. Das hat zur Folge, dass das Eingangssignal mit 6° Auflösung von der Winkeluhr elektronisch verfeinert werden muss,

- bedingt durch die Lücken im Geberrad fehlen an bestimmten Positionen einige dieser Flanken. Zudem sind die Flanken vor und nach einer Lücke etwas verschoben.
- Die mechanische Genauigkeit der Zähne und Lücken ist durch Fertigungstoleranzen mit durchaus relevanten Fehlern behaftet. Diese Fehler können zukünftig z.B. durch eine Geberradkompensation in der rekonfigurierbaren Logik kompensiert werden.

6.3.1.2 Konfigurationsmöglichkeiten für unterschiedliche Geberräder

Neben den Sensorsignalen der Kurbelwelle (Inkrementgeber) und der Nockenwelle (Phasengeber) benötigt die Winkeluhr noch weitere Informationen über den genauen Aufbau der verwendeten Geberräder. Weiterführende Informationen zu Geberrädern und Sensoren zur Position- und Drehzahlerfassung sind im Anhang zu finden.

Damit die Winkeluhr ohne Änderung der rekonfigurierbaren Logik an unterschiedliche Inkrement- und Phasengeberräder angepasst werden kann, wurden nachfolgend beschriebene Konfigurationsmöglichkeiten implementiert.

Bei der Kurbelwelle (KW) kann bei der derzeitigen Implementierung zwischen zwei unterschiedlichen Geberrädern gewählt werden:

- Inkrementgeberrad mit 60-2 Zähnen
- Inkrementgeberrad mit 60-2x2 Zähnen

Da es bei den Nockenwellengeberrädern sehr unterschiedliche Zähnezahlen, Zahnbreiten und weitere Festlegungen gibt, welche Flanken ausgewertet werden müssen, ist hier eine weit reichende Konfigurationsmöglichkeit erforderlich. Deshalb wurden folgende Konfigurationsmöglichkeiten implementiert. Es werden zwei jeweils 120 Bit breite Konfigurationsregister zur Verfügung gestellt. Eines für die fallenden - und eines für die steigenden Flanken des Phasensignals. Die Anzahl von 120 wurde gewählt, da sich bei einer Umdrehung der Nockenwelle die Kurbelwelle genau zweimal dreht und somit 120 Zähne der Kurbelwelle passieren würden. Die Flanken der Nockenwelle werden also immer in Fenstern, die genau die Breite von Zahn zu Zahn der Kurbelwelle haben, betrachtet. Das ist kein zwingender Zusammenhang, stellt aber einen guten Kompromiss zwischen Genauigkeit und Ressourcenverbrauch im FPGA dar.

Im Folgenden wird das Prinzip an einem Beispiel mit weniger Zähnen erläutert (Abbildung 6-53). Betrachtet man zunächst das Konfigurationsregister für die steigenden Flanken. Mit jeder logischen Eins wird damit ein Fenster (rosa) aufgespannt, innerhalb dessen eine steigende Flanke erwartet wird. Tritt innerhalb eines jeden dieser Fenster genau eine steigende Flanke auf, so wird diese als gültig erkannt. Tritt aber einer der folgenden Fälle ein, so wird von der Winkeluhr ein Fehler erkannt und das SYNCHRON Signal deaktiviert:

- Es kommt eine Flanke außerhalb des Fensters,
- es kommt keine Flanke innerhalb des Fensters,
- es kommen mehrere Flanken innerhalb des Fensters.

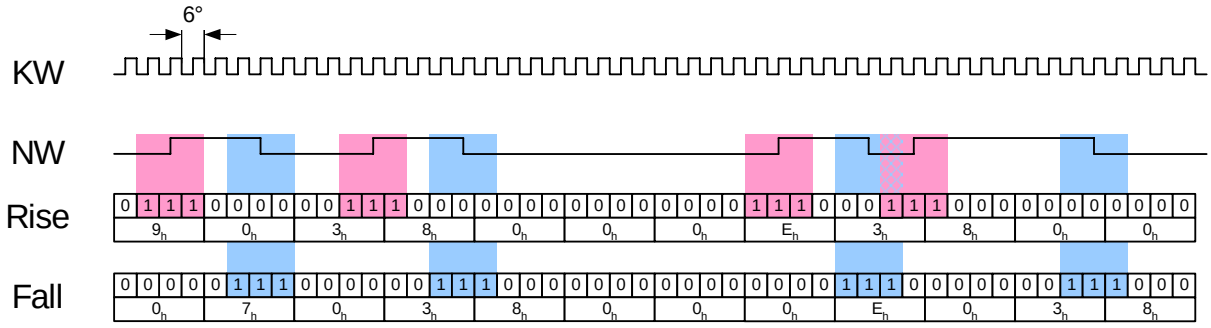


Abbildung 6-53: Konfigurationsmöglichkeiten Nockenwelle

Dasselbe erfolgt für die fallende Flanke (blaues Fenster).

Es ist nicht immer erforderlich oder erwünscht, sowohl die steigende, als auch die fallende Flanke auszuwerten. Aus diesem Grund wurde eine weitere Konfigurationsmöglichkeit implementiert. Dabei kann für jedes Segment unabhängig festgelegt werden, welche Flanken ausgewertet werden sollen. Derzeit werden 4 Segmente unterstützt, was zu vier Konfigurationsregistern mit jeweils zwei Bit führt:

- EDGEx_EN [1:0] = „00“ → none,
- EDGEx_EN [1:0] = „01“ → rising,
- EDGEx_EN [1:0] = „10“ → falling,
- EDGEx_EN [1:0] = „11“ → both.

6.3.2 Ansteuerung von Einspritzventilen

Die Genauigkeit, mit welcher der Einspritzbeginn und die Einspritzmenge gesteuert werden können, hat einen erheblichen Einfluss auf die Abgas- und Geräuschemission sowie auf den Kraftstoffverbrauch. Um diese Parameter flexibler beeinflussen zu können, werden zunehmend die mechanischen Regelungen durch elektronische ersetzt. Die Serien-systeme sind heute schon in bestimmten Bereichen konfigurier- und beeinflussbar. Um neue Konzepte evaluieren zu können, reicht dies aber häufig nicht aus. In [Bos01] und [Bos02_1] wird ein Überblick über die Funktionsweise, den Aufbau sowie die Leistungs-fähigkeit und die Einsatzgebiete der bereits in Serie befindlichen Einspritzsysteme gegeben.

Die weitere Betrachtung der Einspritzsysteme beschränkt sich auf das magnetventilgesteuerte Einzelzylinder-System (**U**nit **I**njektor **S**ystem) sowie das magnet-ventilgesteuerte Speichereinspritzsystem (**C**ommon **R**ail), da diese bei entsprechender Ansteuerung die größte Flexibilität bieten. Informationen zu weiteren Einspritzsystemen bzw. weitergehende Details können unter anderem in [Bos01] und [Bos02_1] nachge-schlagen werden.

6.3.2.1 Anforderungen an ein flexibles Diesel-Einspritzsystem

Die wesentlichen Eigenschaften eines flexiblen Ansteuersystems für Magnetventil Injektoren sind:

- **Ansteuerung von bis zu 8 Magnetventilen (Zylindern)**
  (Mit einem Serien Steuergerät können meist 6 Magnetventile angesteuert werden. Für 8 und Mehrzylindermotoren werden dann 2 Steuergeräte verwendet. Das parallele Verwenden mehrerer Entwicklungssysteme soll ebenfalls möglich sein),
- **Aufteilung der 8 Zylinder auf 2 unabhängige Bänke**
  (Damit wird eine überlappende Einspritzung in zwei Zylindern ermöglicht),
- **bis zu 5 Einspritzungen je Zylinder und Arbeitsspiel,**
- **Bestromungsbeginn für jede Einspritzung frei konfigurierbar,**
- **Bestromungszeiten für jede Einspritzung frei konfigurierbar,**
- **Stromschwellen in weiten Bereichen frei konfigurierbar,**
- **Ansteuermöglichkeit für UIS Injektoren,**
- **Ansteuerungsmöglichkeit für CR Injektoren,**
- **Flexible Erweiterbarkeit für neue Anforderungen.**

Auf Optimierungen an den Injektoren wird in dieser Arbeit nicht eingegangen.

6.3.2.2 Realisierung eines flexiblen Einspritzsystems

Bei der Realisierung wurde sehr darauf geachtet, die maximale Flexibilität bei größtmöglicher Identität mit dem Seriensystem zu realisieren. Das hat zu folgendem Ansatz geführt:

- Es wird die Leistungselektronik aus dem Seriensystem übernommen (Diese ist auf dem aktuellen Stand der Technik und bietet somit kein größeres Optimierungspotential),
- die Ansteuerung sowie die Signalauswertung, welche im Seriensystem gemeinsam durch die TPU des MPC555 und einen ASIC bereitgestellt werden, wurden in zwei Subsysteme aufgeteilt. Die Ansteuerung wird mit einem FPGA realisiert, wohingegen die analoge Signalauswertung diskret aufgebaut wird.

Der Aufbau ist im folgenden Übersichtsblockdiagramm sowie in den detaillierten Blockdiagrammen der Subsysteme dargestellt.

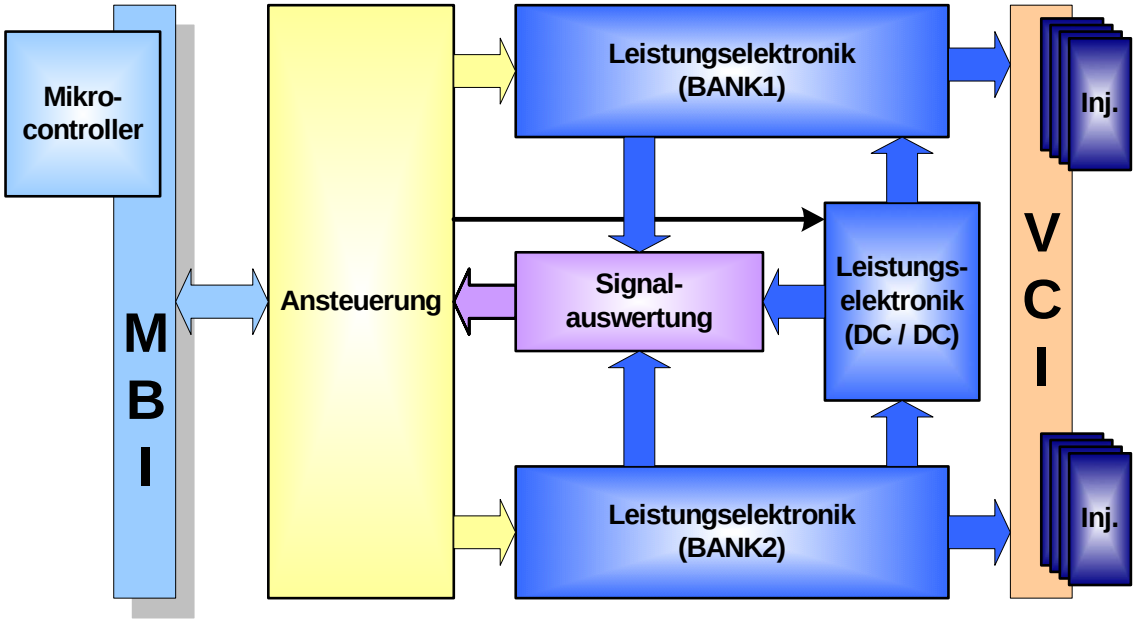


Abbildung 6-54: Übersichtsblockdiagramm Einspritzung

Die Ansteuerung der Leistungsendstufen ist komplett im FPGA realisiert. Damit ist es möglich, ein sehr verzögerungsarmes Verhalten zu implementieren und dennoch sehr flexibel mit Änderungen umgehen zu können. Die für die Ansteuerung relevanten Eingangsgrößen werden zum einen vom Mikrocontroller (MPC555) und zum anderen von der Winkeluhr (Vergl. Abschnitt 6.3) geliefert.

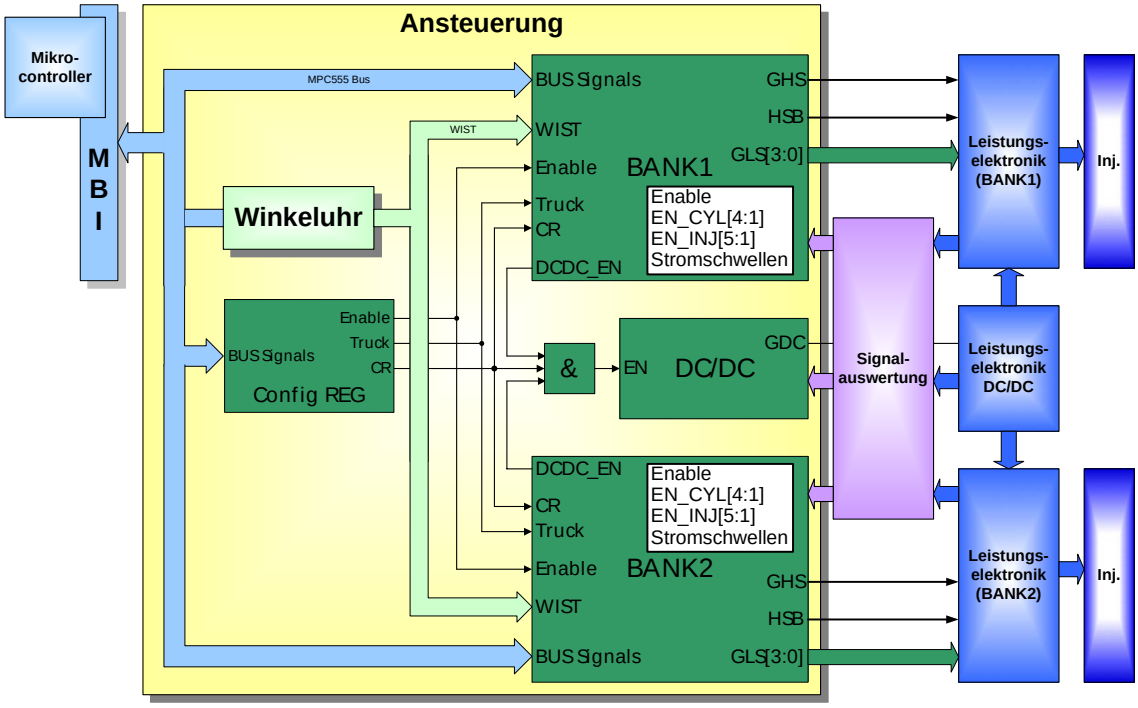


Abbildung 6-55: Blockdiagramm Ansteuerung

Es wurden folgende globale (für beide Bänke gemeinsame) Konfigurationsmöglichkeiten implementiert:

- **Enable**
  Damit ist es möglich, beide Bänke gleichzeitig ein bzw. auszuschalten,
- **CR / UIS**
  Umschaltung zwischen dem Stromkurvenverlauf für CR (vergl. Abschnitt 6.3.2.3) und UIS (vergl. 6.3.2.4),
- **TRUCK**
  Umschaltung zwischen dem Stromkurvenverlauf für TRUCK und CAR.

In Abbildung 6-56 ist die Leistungselektronik einer Bank dargestellt. Die zweite Bank ist identisch aufgebaut.

In der oberen Hälfte befinden sich zwei High Side Schalter. Diese Schalter wirken jeweils für alle vier Zylinder einer Bank gemeinsam. Mit dem unteren der beiden Schalter kann kurzzeitig eine Boost Spannung mit bis zu ca. 50 V auf die High Side Ausgänge geschaltet werden. Mit dem oberen Schalter kann die Batteriespannung (V_V_UBR) auf die High Side Ausgänge geschaltet werden. Der Strom, welcher durch den oberen Schalter fließt, kann mit Hilfe des darüber angeordneten Shunt gemessen werden.

In der unteren Hälfte befinden sich die Low Side Schalter. Hier gibt es für jeden Zylinder einen separaten Schalter. Jeder Ausgang verfügt über eine Freilaufdiode. Der Strom kann mit Hilfe des unten angeordneten Shunt gemessen werden.

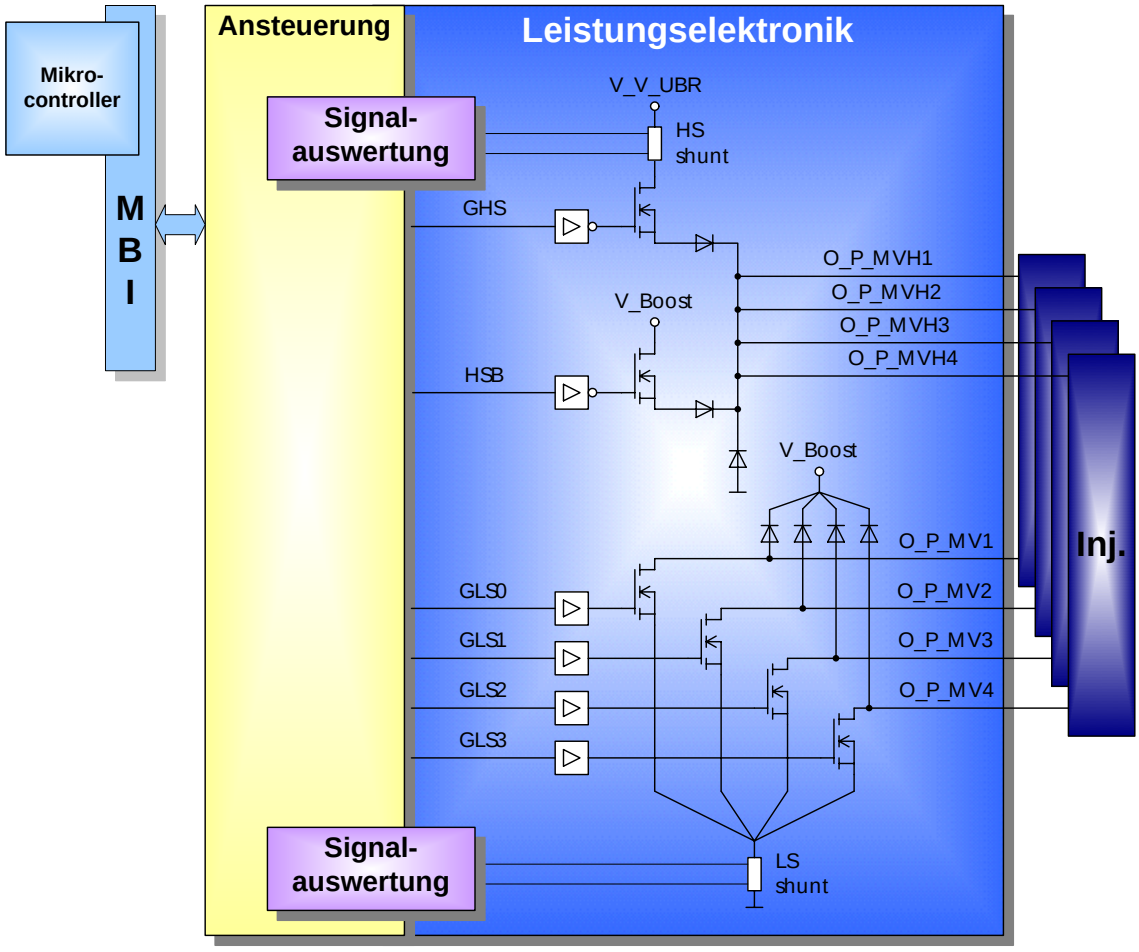


Abbildung 6-56: Blockdiagramm Leistungselektronik einer Bank

In den folgenden Abschnitten wird auf die Ansteuerung von UIS und CR Injektoren genauer eingegangen.

6.3.2.3 Ansteuerung eines Unit Injektor System (UIS)

Die Ansteuerung des Unit Injektors erfolgt in vier Phasen (vergl. Abbildung 6-57):

1. Anzugsstromphase (T_1)
   Hier wird mit einer Zweipunkt Stromregelung auf hohem Stromniveau das Magnetventil, welches sich zwischen Hoch- und Niedrigdruckbereich befindet, relativ schnell in Bewegung versetzt (Stromeinprägung). Die Phase beginnt an einem durch den Mikrocontroller vorgebbaren Kurbelwellenwinkel (A_{Start}) und endet nach der Zeit (T_1), welche ebenfalls durch den Mikrocontroller konfiguriert werden kann. Die Zeit (T_1) wird so vorgegeben, dass diese Phase endet, bevor das Magnetventil vollständig geschlossen ist (Aufschlag auf Ventilsitz).

2. BIP Erkennungsphase (T_2)
   In dieser Phase findet keine Stromregelung statt, damit der Aufschlag auf den Ventilsitz und damit quasi der Einspritzbeginn (**B**egin of **I**njection **P**oint, BIP) detektiert werden kann. Der Aufschlag auf den Ventilsitz kann an einem Knick im Stromverlauf erkannt werden. Die Dauer (T_2) und damit das Ende dieser Phase wird ebenfalls durch den Mikrocontroller vorgegeben.

3. Haltestromphase (T_3)
   In dieser Phase (zuzüglich der Zeit von BIP Erkennung bis Ende (T_2)) findet die Einspritzung statt. Hier wird mit einer Zweipunkt Stromregelung das Ventil geschlossen gehalten. Das Halten kann auf einem niedrigeren Stromniveau als bei der Anzugsstromphase erfolgen.

4. Schnelllöschphase
   Hier wird durch Schnelllöschung das Magnetventil so schnell wie möglich geöffnet. Damit wird erreicht, dass der Druck im Hochdruckbereich zusammenbricht und somit das Einspritzende relativ genau vorgegeben werden kann.

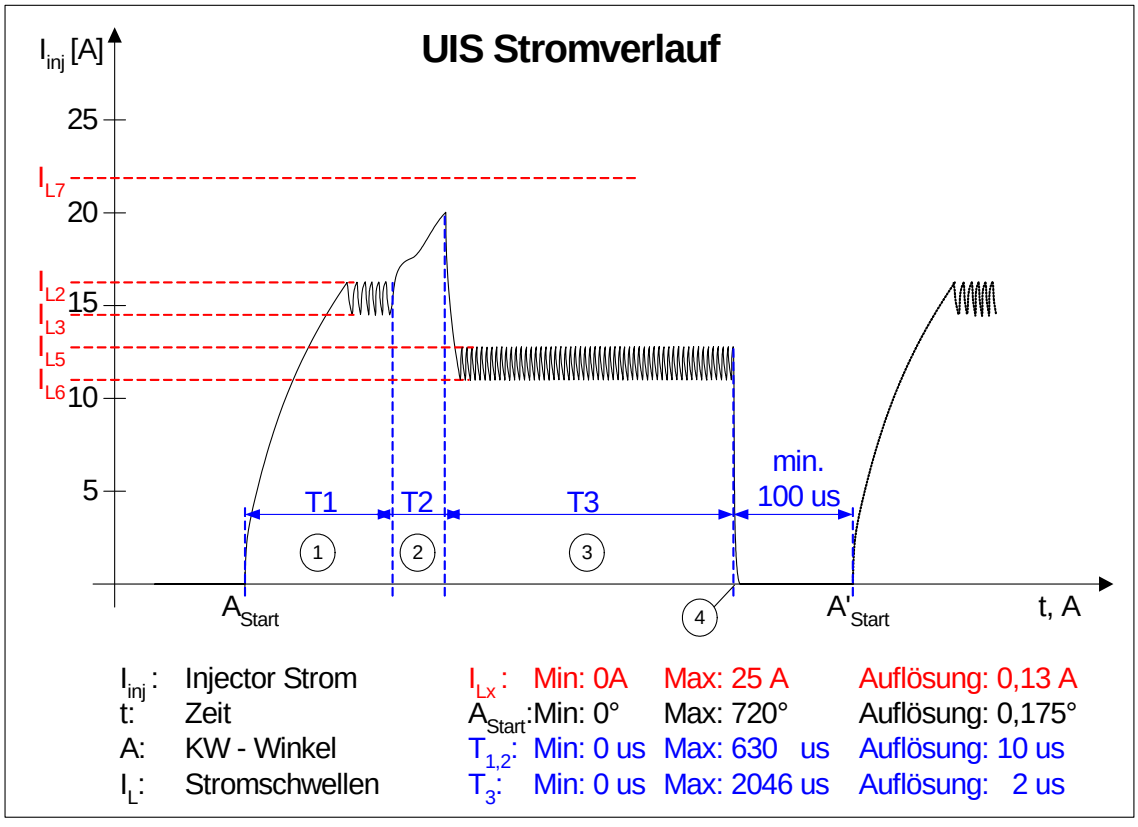


Abbildung 6-57: Stromverlauf UIS Einzeleinspritzung

Fünf solcher Einzeleinspritzungen können je Zylinder und Arbeitsspiel maximal realisiert werden (vergl. Abbildung 6-58). Es wäre möglich, diese Anzahl an Einspritzungen zu erhöhen, doch ist es mit derzeit verfügbaren Injektoren nicht möglich, dieses dann auch real anzuwenden.

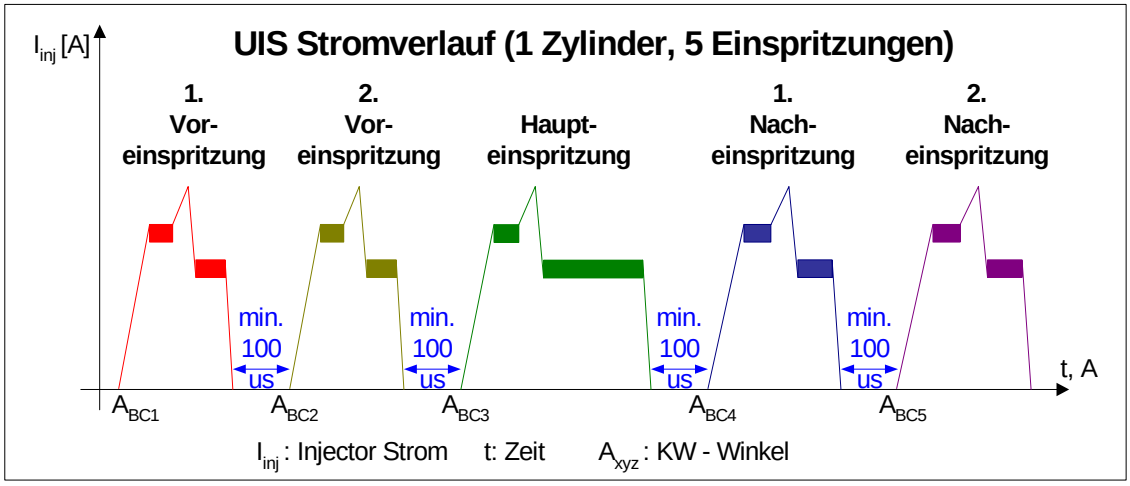


Abbildung 6-58: Stromverlauf UIS 5 Einspritzung je Zylinder

Die Pausezeiten müssen auch zwischen zwei Einspritzungen unterschiedlicher Zylinder eingehalten werden, da pro Bank nur ein High Side Mosfet existiert und dessen Treiber aufgeladen werden muss.

Die zwei Bänke beim Zweibankbetrieb hingegen sind unabhängig voneinander. Die Einspritzungen der beiden Bänke dürfen sich daher überlappen.

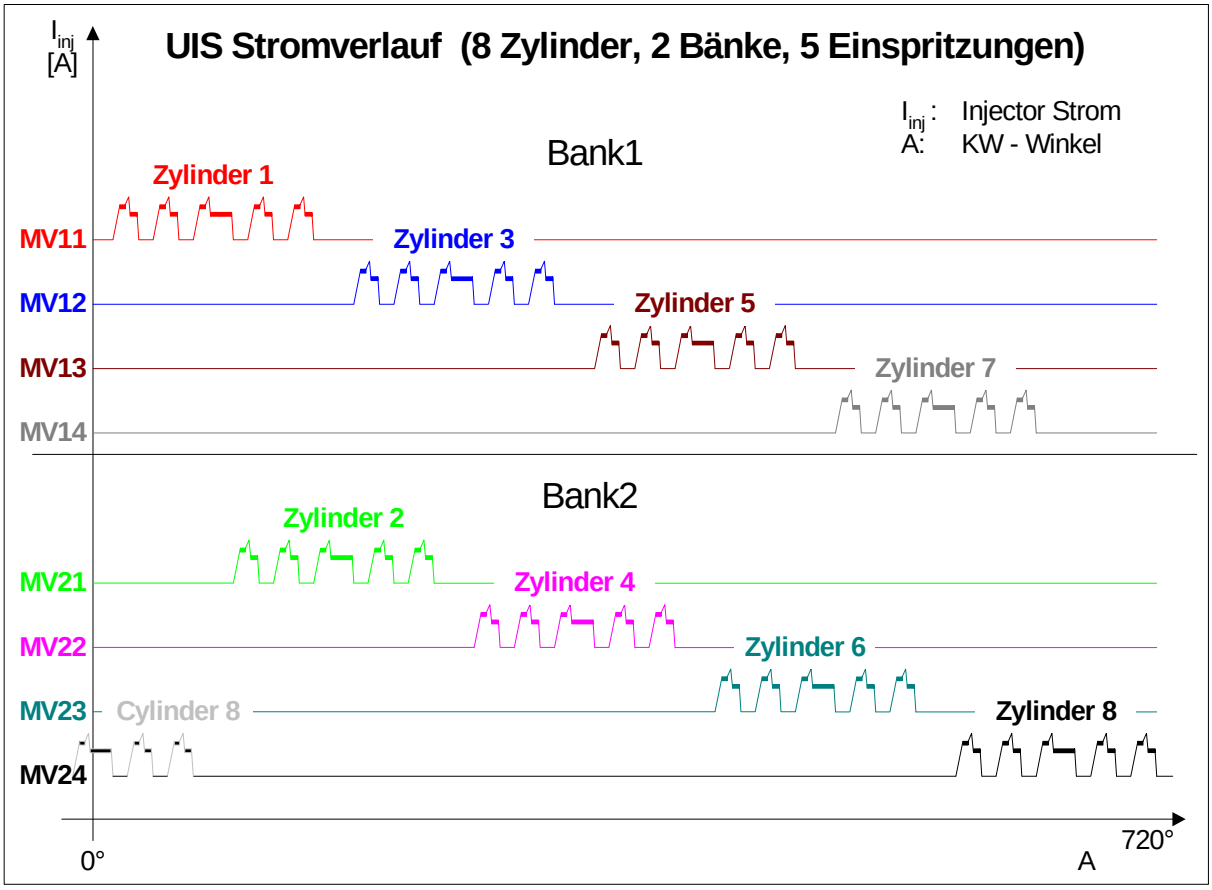


Abbildung 6-59: Stromverlauf UIS Einspritzung für 2 Bänke

Neben den oben bereits erwähnten Parametern für Bestromungsbeginn und den Zeiten für die einzelnen Phasen können noch weitere Einstellungen vorgenommen werden:

- **Enable**
  Damit ist es möglich, jede Bank separat ein- bzw. auszuschalten,
- **EN_CYL[4:1]**
  Freigabe einzelner Zylinder. Für jeden Zylinder einer Bank ist ein Bit vorhanden, mit dessen Hilfe der Zylinder aktiviert bzw. deaktiviert werden kann,
- **EN_INJ[5:1]**
  Freigabe einzelner Einspritzungen. Für jede der 5 Einspritzungen ist ein Bit vorhanden, mit dessen Hilfe die Einspritzung für alle Zylinder dieser Bank aktiviert bzw. deaktiviert werden kann. Zudem ist es möglich, durch Nullsetzen der (T_1) einer spezifischen Einspritzung diese zu deaktivieren,
- **Stromschwellen**
  Es können sämtliche Stromschwellen konfiguriert werden.

Die Stromschwellen werden nur beim Systemstart ausgewertet. Während des Betriebs bleiben diese konstant. Alle anderen Werte können im Betrieb geändert werden. Dabei sind allerdings einige Randbedingungen zu beachten (vergl. Abbildung 6-60). Die Parameter dürfen nicht geändert werden so lange die Einspritzung läuft sowie 100 µs bevor sie beginnen. Ob ein Parameter aktuell beschrieben werden darf, kann über ein Hardware Register abgefragt werden. Diese Abfrage ist fester Bestandteil der untersten Softwareschicht.

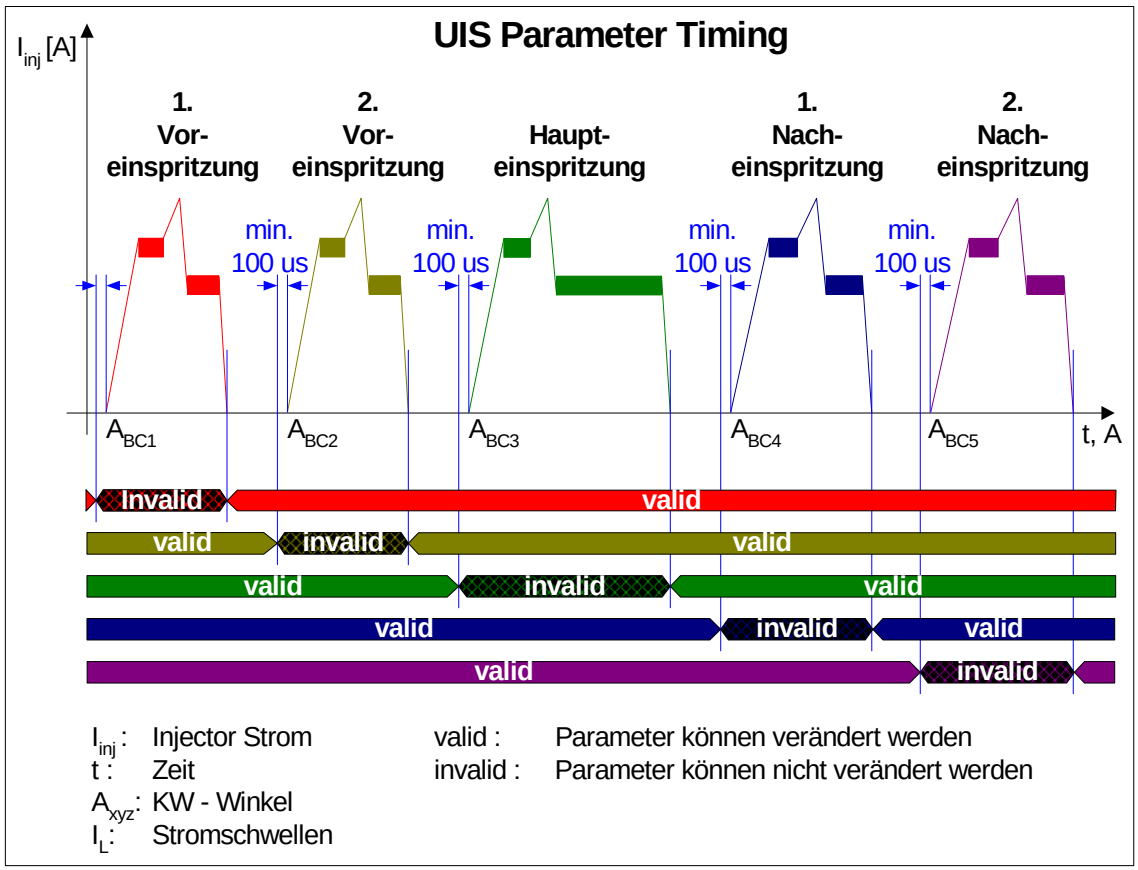


Abbildung 6-60: Timing für Variation der UIS Einspritzparameter

6.3.2.4 Ansteuerung eines Common Rail (CR) Systems

Die Ansteuerung des CR Injektors erfolgt in vier Phasen (vergl. Abbildung 6-61):

1. Beschleunigungsphase (A_{Start} bis I_{L1})
   Die Phase beginnt an einem durch den Mikrocontroller vorgebbaren Kurbelwellenwinkel (A_{Start}). Zum schnellen und reproduzierbaren Öffnen des Magnetventils wird zunächst ein Strom mit einer sehr steilen Flanke eingeleitet. Dieser steile Stromanstieg wird mit einer Boosterspannung von ca. 50 V erreicht. Diese Phase endet, sobald der Strom durch den Injektor die Größe I_{L1} erreicht hat.

2. Anzugsstromphase (I_{L1} bis Ende T1)
   In dieser Phase wird mit einer Zweipunkt Stromregelung auf hohem Stromniveau das Magnetventil relativ schnell weiter bewegt. Als Versorgungsspannung wird hier die Batteriespannung (V_V_UBR) verwendet. Die Phase endet nach der Zeit (T_1), welche ebenfalls durch den Mikrocontroller konfiguriert werden kann.

3. Haltestromphase (T_3)
   In dieser Phase wird ebenfalls mit einer Zweipunkt Stromregelung das Ventil offen gehalten. Das Halten kann auf einem niedrigeren Stromniveau als bei der Anzugsstromphase erfolgen.

4. Schnelllöschphase
   Hier wird durch Schnelllöschung das Magnetventil so schnell als möglich geschlossen. Damit wird erreicht, dass das Einspritzende relativ genau vorgegeben werden kann. Die Energie, welche beim Schnelllöschen frei wird, wird dazu verwendet, die Boosterkondensatoren aufzuladen. Die fehlende Energie, um die Kondensator auf die volle Boosterspannung zu laden, wird in den Pausezeiten von einem DC/DC Wandler zur Verfügung gestellt.

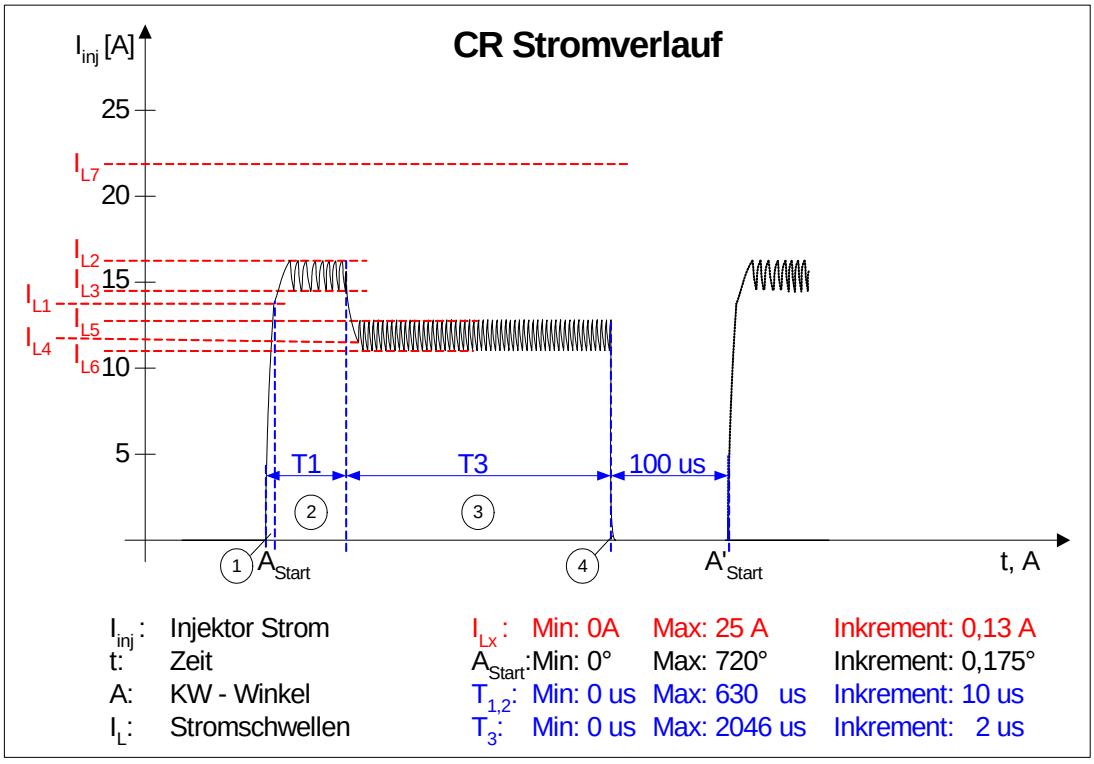


Abbildung 6-61: Stromverlauf CR Einzeleinspritzung

Zur Konfiguration stehen analog zum UIS System mit Ausnahme von T_2 auch hier dieselben Parameter zur Verfügung.

6.3.3 Interrupts für ereignisgesteuerte Aufgaben

Bei der Steuerung und Regelung von Verbrennungsmaschinen werden neben zeitgesteuerten auch ereignisgesteuerte Software-Tasks benötigt. Das wichtigste Beispiel für ereignisgesteuerte Software-Tasks sind kurbelwinkelsynchrone Berechnungen. Hier müssen beispielsweise rechtzeitig zur nächsten Einspritzung die entsprechenden Parameter zur Verfügung gestellt werden. Übliche Zeiten bzw. Winkel sind 1 ms, 10 ms, 100 ms und z.B. 180° KW.

6.3.3.1 Interrupts auf dem Mikrocontroller (MPC555)

Der Mikrocontroller (MPC555) verfügt in Summe über 16 Interrupts. Acht davon werden von internen Quellen, wie z.B. Timer oder A/D-Wandler gespeist. Diese internen Interrupts werden auch als Level[7:0] Interrupts bezeichnet. Die verbleibenden Acht sind nach außen geführt. Diese werden üblicherweise als IRQ[7:0] bezeichnet. Alle diese Interrupts werden im Controller auf einen einzigen Interrupt abgebildet, was dazu führt, dass beim Auftreten eines Interrupts zuerst bestimmt werden muss, welche der Quellen ihn ausgelöst hat. Die 16 Interrupts besitzen unterschiedliche Prioritäten (0 hat die höchste und 15 die niedrigste). Wobei sich externe und interne Interrupts beginnend mit dem IRQ[0] gefolgt von Level[0] jeweils abwechseln. (Abbildung 6-62)

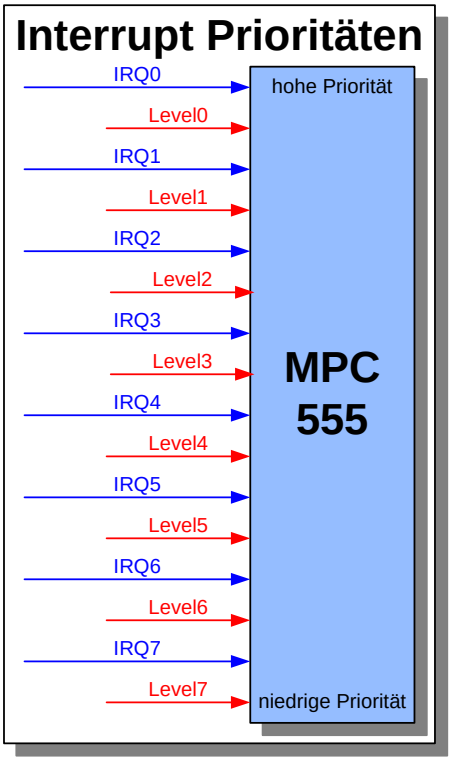


Abbildung 6-62: Interrupt Prioritäten

Auf die internen Interrupts wird im Folgenden nicht näher eingegangen, ihre Funktionsweise kann in [Mot00] nachgeschlagen werden.

Die externen Interrupts IRQ[7:3] der ES1640 können über den VMEbus ausgelöst werden. Die verbleibenden IRQ[2:0] werden vom FPGA oder von Weiteren an das ABI angeschlossenen Systemen kontrolliert. (Abbildung 6-63)

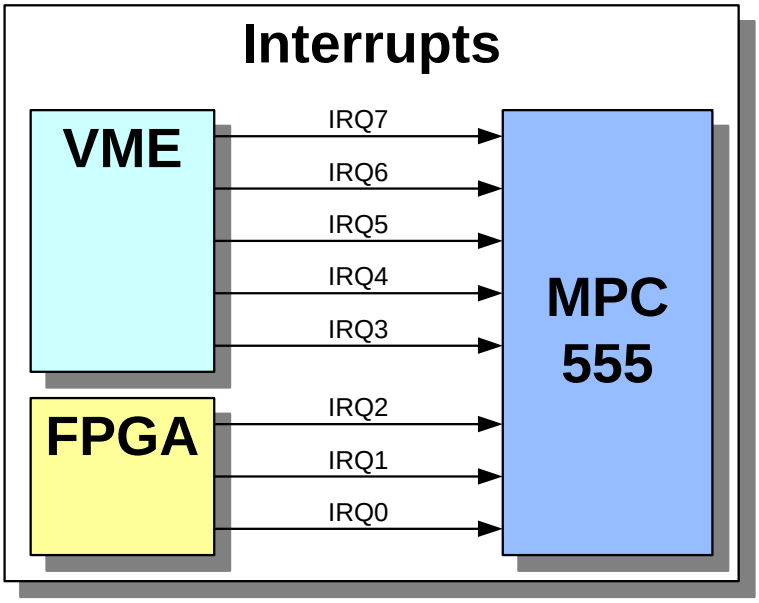


Abbildung 6-63: Interrupteingänge des MPC555

6.3.3.2 Realisierung eines flexiblen Interrupters

Um bei der Interruptgenerierung flexibel zu sein, wurde der Interrupter im FPGA implementiert. Es wurde ein 16-fach Interrupter realisiert. Das bedeutet, dass bis zu 16 unterschiedliche Interrupts ausgelöst werden können. Alle diese 16 Interrupts werden zu einem einzigen zusammengefasst. Über ein Konfigurationsregister kann festgelegt werden, auf welchen der 3 freien IRQs des MPC555 dieser geführt werden soll.

Über das IRQ_EN Register kann der komplette Interrupter freigegeben bzw. gesperrt werden.

Mit dem IRQ_SRC_EN[15:0] kann jeder einzelne Interrupt freigegeben bzw. gesperrt werden.

Ist ein Interrupt freigegeben und wird eine steigende Flanke am IRQ_SRC Eingang erkannt, so wird der Interrupt im entsprechenden IRQ Register gespeichert und gleichzeitig an den ausgewählten Interrupteingang des MPC555 signalisiert (vergl. Abbildung 6-64). Die Interrupt Serviceroutine des MPC555 kann die IRQ[15:0] Register auslesen und somit feststellen, wer den Interrupt ausgelöst hat. Durch Beschreiben der IRQ Registers durch den MPC555 wird dieses zurückgesetzt und der Interrupt somit gelöscht.

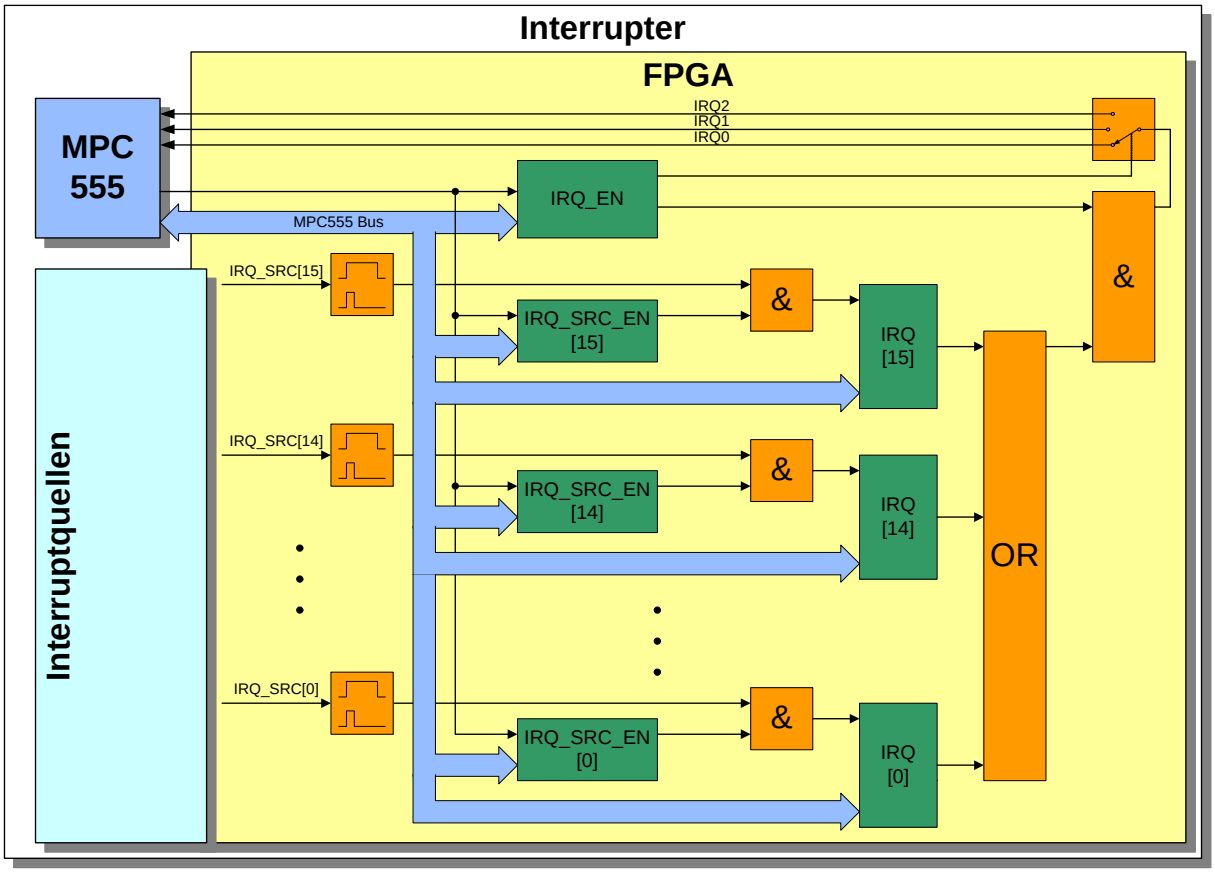


Abbildung 6-64: Interrupter für MPC555

Acht Eingänge des Interrupters wurden zur Generierung von Kurbelwellenwinkel-synchronen Interrupts ausgelegt. Die restlichen acht sind derzeit unbenutzt. Diese können bei Bedarf aktiviert werden. In die Register A1 bis A8 (vergl. Abbildung 6-65) kann der MPC555 jeweils einen Winkel schreiben, bei dem ein Interrupt generiert werden soll. Ein Komparator vergleicht ständig diese Register mit der aktuellen Kurbelwellenposition. Stimmen die beiden Winkel überein, so wird der entsprechende Interrupt ausgelöst.

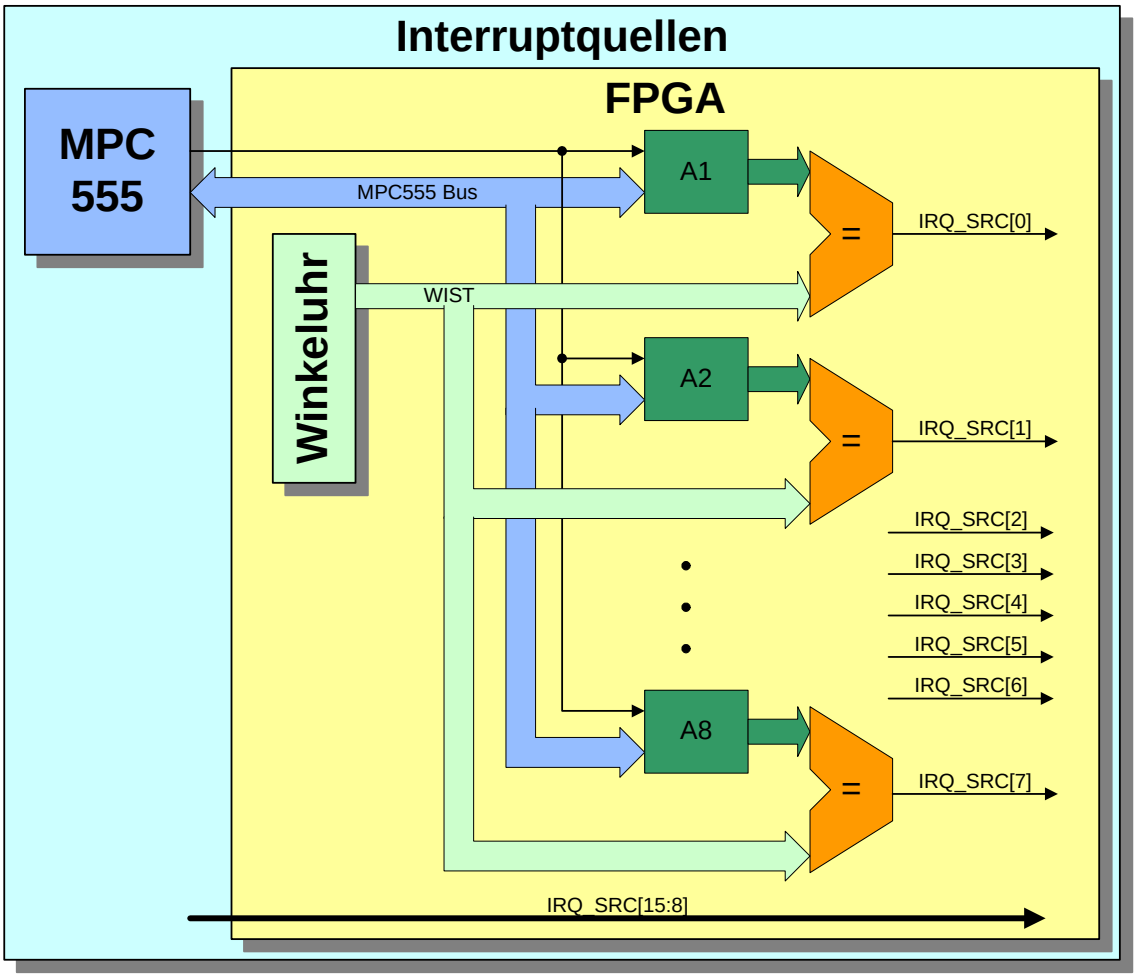


Abbildung 6-65: Interruptquellen MPC555

Kapitel 7

ZUSAMMENFASSUNG

7 Zusammenfassung

In der vorliegenden Arbeit wird ein zielsystemidentischer Ansatz für das domänenspezifische Rapid Prototyping in der Informations- und Elektrotechnik vorgestellt, sowie dessen Machbarkeit am Beispiel eines Entwicklungssystems für die Elektronik im Kraftfahrzeug nachgewiesen.

Durch das hier vorgestellte domänenspezifische Rapid Prototyping in der Informations- und Elektrotechnik wird es ermöglicht, sehr schnell neue Konzepte und Anforderungen umzusetzen und in der realen Umgebung auf Funktion und Akzeptanz zu testen. Zusätzlich können die erarbeiteten Ergebnisse ohne große Änderungen in das spätere Zielsystem übernommen werden. Ein Schwerpunkt dieser Arbeit ist es, die bislang den Systemlieferanten vorbehaltene Wiederverwendung von Erfahrungen und Ergebnissen aus Serienentwicklungen, systematisch voranzubringen und gut handhabbar für Forschungs- und Entwicklungsaufgaben zur Verfügung zu stellen. Mit diesem Ansatz werden diese Erfahrungen und Ergebnisse unter anderem den OEM sowie Spezialfirmen zugänglich, ohne dass die Systemlieferanten ihr komplettes Know-How offen legen müssen. Damit werden die OEM in die Lage versetzt, zukünftige Innovationen selbst im eigenen Unternehmen voranzutreiben und somit früher als ihre Wettbewerber mit neuen Ideen in den Markt zu gehen. Darüber hinaus werden beispielsweise sehr effiziente Möglichkeiten für das Simultaneous Engineering zwischen OEM, Systemlieferant und Spezialfirma eröffnet. Durch diese Wiederverwendung in einem Entwicklungssystem kann aber nicht nur dieses Entwicklungssystem schneller dargestellt werden, sondern es ist auch deutlich einfacher möglich, die mit diesem System dargestellten Lösungen erneut in eine Serienlösung zu überführen. Der frühe und konsequente Einsatz von Standards der Softwareentwicklung der Fahrzeugindustrie (wie beispielsweise CARTRONIC, OSEK/VDX und ASAM) ist dabei ein wichtiger Aspekt, um die Wiederverwendung zu ermöglichen. Die Steigerung des Automatisierungsgrades von Entwicklungsprozessen sowie der Einsatz der automatischen Codegenerierung reduziert Fehlerquellen und beschleunigen die Entwicklung.

Die im ersten Abschnitt dargestellten ständig steigenden Anforderungen von Gesetzgeber und Kunden an die Kosten, Sicherheit, Umweltverträglichkeit sowie den Komfort neuer Produkte, erfordern im Bereich der Fahrzeugtechnik neue Konzepte und Technologien. In diesem Bereich wird die Elektrik, Elektronik und Software die Mechanik und Hydraulik zurückdrängen oder in einer Symbiose ergänzen und damit zur Schlüsseltechnologie im Fahrzeugbau aufsteigen [Mer02]. Das Beherrschen dieser neuen Technologien sowie die schnelle Erprobung und anschließende Umsetzung in Produkte wird zum Erfolgsfaktor für zukunftsfähige Unternehmen. Als mögliche Maßnahmen diese Anforderungen zu erfüllen, wurden die Wiederverwendung von Systemteilen aus Serienprojekten, die Reduktion von Doppel- und Parallelentwicklungen sowie die Erhöhung des Automatisierungsgrades identifiziert. Als zielführende Technologie eignen sich vor allem eingebettete mikroelektronische sowie mechatronische Ansätze. Gestützt durch die Ergebnisse der Mercer Studie [Mer02] konzentriert sich diese Arbeit auf die Optimierung des domänenspezifischen Entwurfs im Bereich Elektrotechnik und Informationstechnik. Dabei stehen die schnelle und einfache Realisierung eines Labormusters bzw. eines Funktionsmusters im Vordergrund. Damit die Fahrzeughersteller zukünftige Innovationen selbst oder mit unabhängigen Spezialfirmen vorantreiben und somit früher als ihre Wettbewerber mit neuen Ideen in den Markt gehen können, wird eine neue Aufteilung der Kompetenzen angestrebt. Bei diesem Vorgehensmodell wird bewusst auf die Einbindung eines Systemlieferanten bis zur Fertigstellung eines Konzepts verzichtet. Durch die direkte Zusammenarbeit mit Firmen, welche spezielles Know-How bezüglich des Einsatzes bestimmter Technologien haben, wird nach der für den Hersteller besten und günstigsten Lösung gesucht. Nachdem die Entscheidung für ein bestimmtes Konzept gefallen ist, wird ein Systemlieferant ausgewählt und die Verantwortung für die weitere Entwicklung an diesen übertragen.

Mit Hilfe der im vierten Abschnitt analysierten und bewerteten Entwurfsmethoden für eingebettete Systeme (Rapid Prototyping, Bypass) wurde aufgezeigt, welcher Ansatz für welche Aufgabenstellung am besten geeignet ist und in welchem Bereich noch Möglichkeiten für Optimierungen bestehen. Zudem wurde dargestellt, von welchem Entwicklungspartner welcher Aufwand bei den unterschiedlichen Ansätzen erbracht werden muss. Abbildung 7-1 zeigt die Vorteile des domänenspezifischen Rapid Prototyping gegenüber einem Universal Experimentiersystem sowie internem und externem Bypass bezüglich Flexibilität, Performance und Identität mit dem Zielsystem.

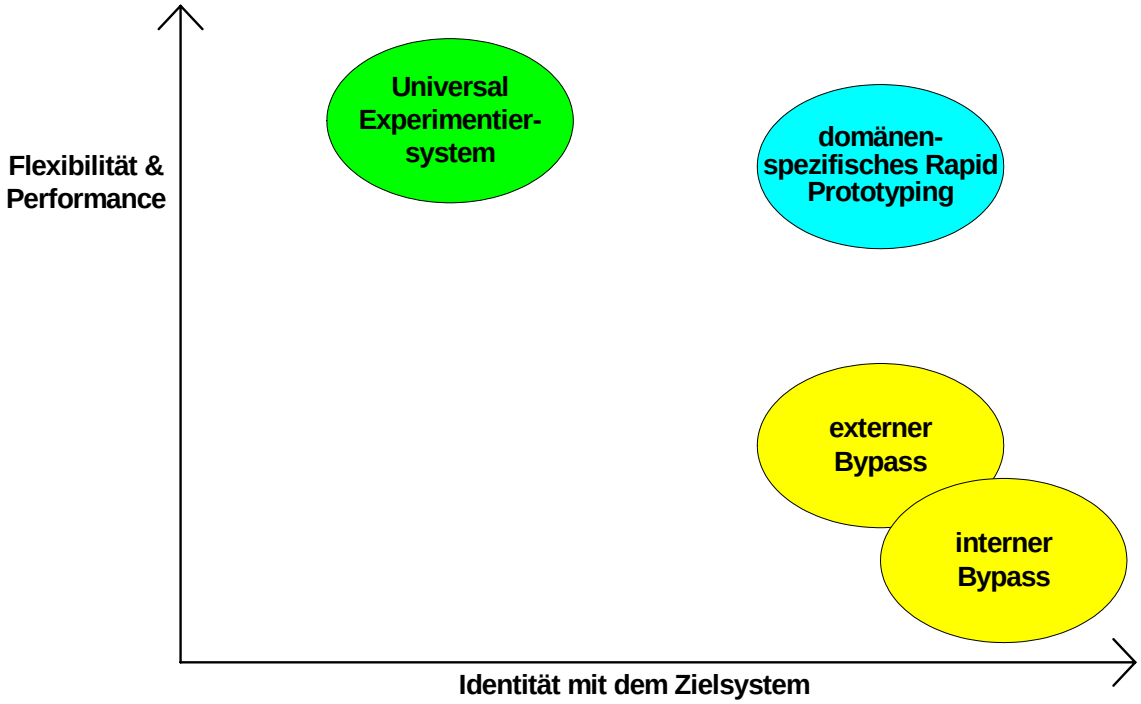


Abbildung 7-1: Vergleich verschiedener Entwicklungsmethoden

Das im fünften Abschnitt vorgestellte Konzept zur Verkürzung der Entwicklungszeit komplexer Systeme, nutzt konsequent Erfahrungen aus bereits laufenden Serienprojekten in Form von Simulationsmodellen, Hardware- und Softwarekomponenten. Das Gesamtkonzept besteht aus einem informationstechnischen und einem elektrotechnischen Teil. Der informationstechnische Teil umfasst die Software- und Kommunikationsstruktur des aus einem Simulationsprozessor und einem Mikrocontroller bestehenden Entwicklungssystems. Besonderes Augenmerk wird dabei auf die einfache Übertragbarkeit der Anwendersoftware zwischen den beiden Rechnern gelegt. Der elektrotechnische Teil behandelt die Partionierung der Hardware sowie die damit verbundene Einführung erforderlicher Schnittstellen. Im Bereich der Signalgenerierung und -auswertung wird die Flexibilität gesteigert, ohne die Identität mit dem angestrebten Zielsystem zu verlieren. Das wird erreicht, indem weitgehend auf ASIC's und andere Halbleiter aus Seriensystemen zurückgegriffen wird. Diese werden über eine leistungsfähige **R**ekonfigurierbare **L**ogik (RL) an den Mikrocontroller gekoppelt. Somit ist es sehr schnell möglich, Konfigurationen vergleichbar mit Seriensystemen herzustellen. Weitergehende Anforderungen, wie beispielsweise neue Protokolle für die Kommunikation, spezielle Signalauswertungen (Drehzahl-, Positionserfassung, neue Sensoren, etc.) oder Signalgenerierungen (Einspritzsteuerung, Ventilansteuerung, etc.), können einfach und flexibel in der RL realisiert werden. Die mit Hilfe einer Hardwarebeschreibungssprache beschriebenen und mit der rekonfigurierbaren Logik umgesetzten und verifizierten Lösungen können effizient in Serienlösungen, z.B. mittels ASIC's, überführt werden.

Im sechsten Abschnitt wird eine Realisierung des Konzepts anhand eines Diesel Entwicklungs-Steuergerätes dargestellt. Hierfür wurde sowohl die Hardware, als auch ein Basisstand für die Software entwickelt. Sowohl bei der Hardwareentwicklung, als auch bei der Softwareentwicklung, wird soweit sinnvoll und möglich auf Grundfunktionalitäten der Serienentwicklung von Motorsteuergeräten der Firma BOSCH zurückgegriffen. Bei der Hardware sind das in der Serie erprobte diskrete Schaltungsteile, Bausteine bzw. ASIC's. Zu den Grundfunktionalitäten der Software gehören unter anderem die Services, die Hardwaretreiber sowie das Echtzeitbetriebsystem.

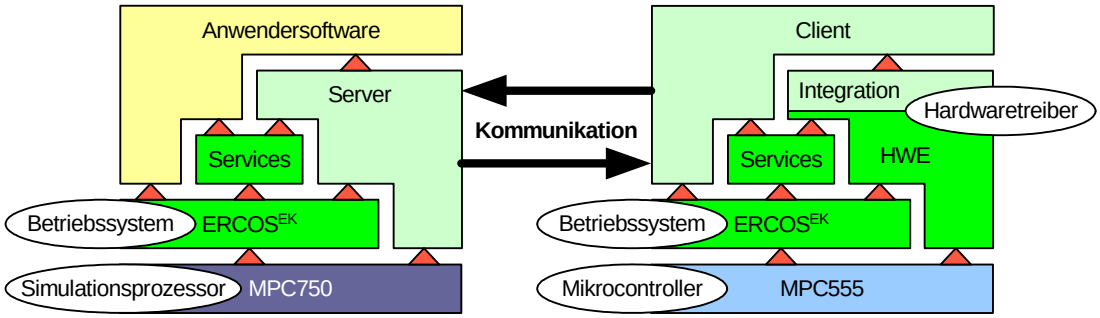


Abbildung 7-2: Softwarestruktur des Entwicklungssystems

Die Integration der Serien I/O Treiberschichten (HWE) sowie der Services in das Softwareentwicklungswerkzeug ASCET wurde auf Klassenbasis realisiert. Des Weiteren wurde eine flexible und leicht zu bedienende Kommunikationsschnittstelle zu einem leistungsfähigen Simulationsprozessor implementiert. Diese auf dem VMEbus basierende Schnittstelle bietet gegenüber der aktuell verfügbaren 100 Mbit Ethernet Schnittstelle des ETK eine deutliche Steigerung der Performance und des Komforts. Die deutlich kürzeren Übertragungszeiten der VME Kommunikation im Vergleich zur bestehenden ETK Kommunikation ist in folgenden Abbildungen zusammenfassend dargestellt. In Abbildung 7-3 sind die Übertragungszeiten vom Mikrocontroller (ES1640) zum Simulationsprozessor (ES1130) dargestellt. Die Abbildung 7-4 zeigt die Übertragungszeiten vom Simulations-prozessor zum Mikrocontroller.

Die Rechenleistung des Entwicklungssystems ist durch die effiziente Möglichkeit einen Simulationsrechner an den Mikrocontroller anzukoppeln sehr einfach skalierbar. Die schnelle und verzögerungsarme Anbindung des Simulationsrechners an den Mikro-controller ermöglicht sehr flexibel die schnelle Realisierung horizontaler Prototypen. Unter Einbeziehung der rekonfigurierbaren Logik sowie seriennaher Hard- und Software sind ebenso vertikale Prototypen realisierbar. Die Leistungsfähigkeit der rekonfigurierbaren Logik wurde exemplarisch anhand einer sehr flexiblen Drehzahl- und Positionserfassung sowie der Ansteuerung von elektromagnetischen Einspritzventilen dargestellt. Diese Funktionen wurden in Anlehnung an die Serien I/O Treiberschichten ebenfalls auf Klassenbasis in ASCET eingebunden.

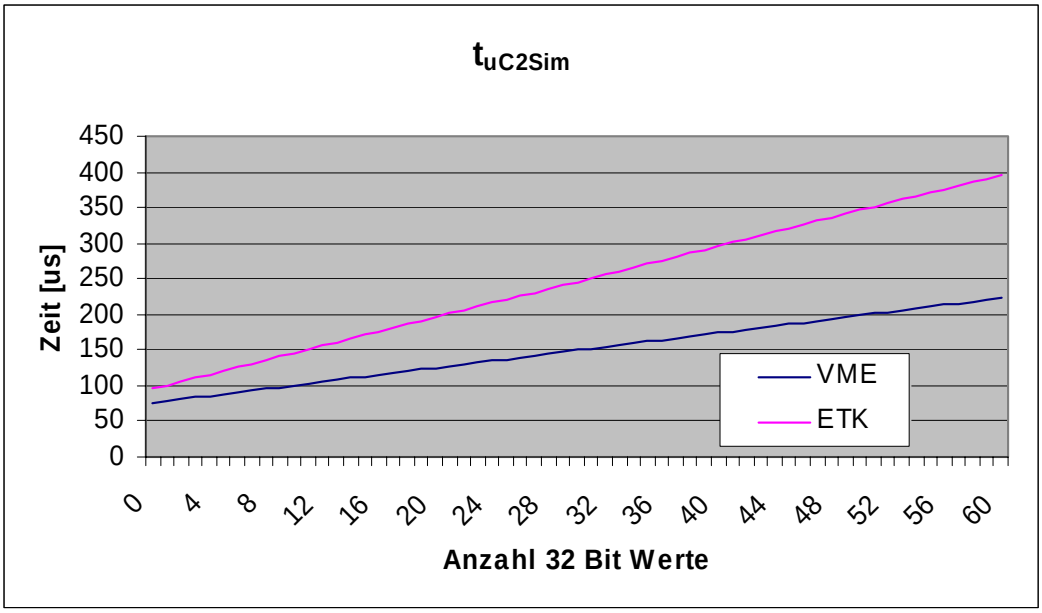


Abbildung 7-3: Übertragungszeiten zum Simulationsprozessor

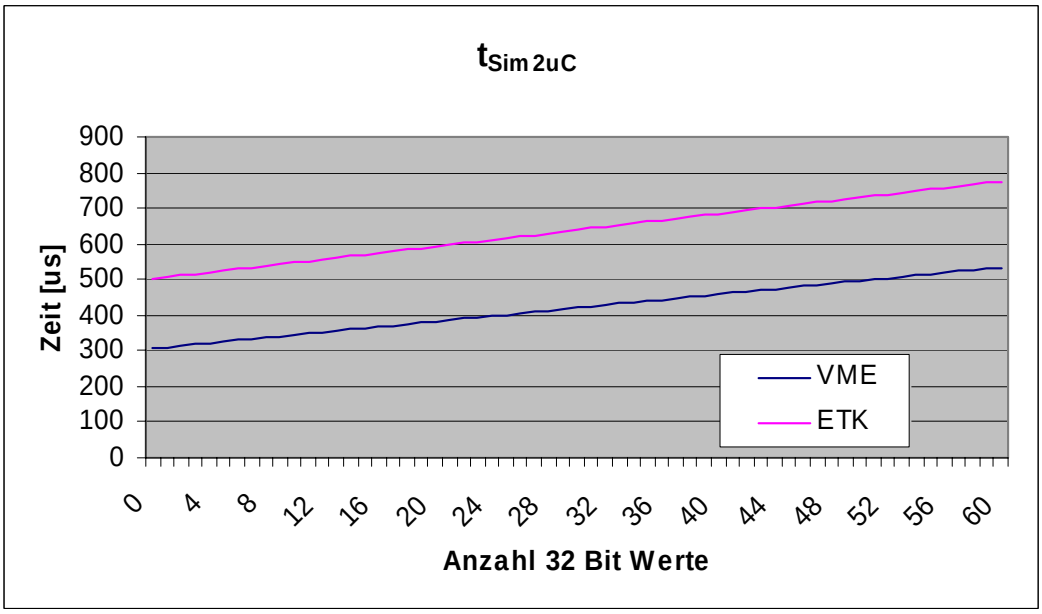


Abbildung 7-4: Übertragungszeiten zum Mikrocontroller

Dieses System wird derzeit gemeinsam mit Forschungs- und Vorausentwicklungs-abteilungen namhafter Pilotkunden der Automobil- und Zuliefererindustrie im Praxiseinsatz getestet.

Nachfolgend sind die Eigenschaften des Systems in Kurzform zusammengestellt:

- **Kurze Entwicklungszeiten**
  Durch die Bereitstellung einer allgemein einsetzbaren Grundfunktionalität sowie durch Wiederverwendung von Erkenntnissen und Komponenten aus der Serienentwicklung ist es sehr schnell möglich, ein Prototypen-System aufzubauen. Mit Hilfe des grafischen Entwicklungswerkzeuges ASCET kann die Formulierung der Lösung mit der am besten passenden Beschreibungsform (Blockdiagramm, Zustandsautomat, textuelle Programmiersprache) sehr schnell vorgenommen und daraus automatisch Code generiert werden.

- **Sehr weit reichende Änderungs- und Erweiterungsmöglichkeiten**
  Änderungen und Erweiterungen können an vielen Stellen sowohl der Hardware (rekonfigurierbare Logik, projektspezifische Bestückvarianten), als auch der Software problemlos und schnell durchgeführt werden. Über die offenen Schnittstellen (VME, EBI) ist es darüber hinaus sehr einfach möglich, Ergebnisse anderer Arbeiten einzubinden.

- **Debugger-Schnittstellen**
  Der Mikrocontroller kann über eine leicht zugängliche BDM-Schnittstelle an einen Debugger angeschlossen werden. Mit dem Debugger kann in der Entwicklungsphase die Software sehr schnell in das RAM des Mikrocontrollers geladen und dort getestet werden. Nachdem die Softwareentwicklung weitgehend abgeschlossen ist, kann über diese Schnittstelle das FLASH des Mikrocontrollers ebenfalls sehr einfach programmiert werden.

  Auf das FPGA kann über die Jag-Schnittstelle zugegriffen werden. Über diese Schnittstelle ist es möglich, den zeitlichen Verlauf interner FPGA Signale zu beobachten.

- **Ausreichende Speicherressourcen (SRAM, FLASH)**
  Der Mikrocontroller wurde extern mit 2 MByte SRAM und 4 MByte FLASH ausgestattet. Das entspricht ca. dem 5- bis 10fachen, was in heutigen Steuergeräten zur Verfügung steht. Der Simulationsprozessor bietet darüber hinaus 64- bis 256 MByte SRAM und 8 MByte FLASH. Damit ist es möglich neue Ideen mit Fokus auf die Funktion zu entwickelt und sobald diese nachgewiesen ist kann die Optimierung auf Speicherressourcen erfolgen.

- **Skalierbare Rechenleistung**
  Die Rechenleistung des Mikrocontrollers (52,7 MIPS [Mot00]) kann einfach mit dem Simulationsprozessor ES1135 (2320 MIPS [IBM03]) erweitert werden. Eine nochmalige Steigerung der Rechenleistung kann durch Einbinden weiterer ES1135 oder neuer leistungsfähigerer Simulationsprozessoren erreicht werden.

- **Betriebssystem identisch mit Zielsystem**
  Es wird das im Automobilbereich weit verbreitete und auf dem OSEK Standard basierende Echtzeitbetriebsystem ERCOS^{EK} verwendet.

- **I/O-Softwaretreiber identisch mit Zielsystem**
  Es werden Softwaretreiber aus modernen Diesel- und Benzinmotor-Steuergeräten so allgemeingültig eingebunden, dass sie auch für andere Bereiche eingesetzt werden können.

- **Prozessor und I/O identisch mit Zielsystem**
  Bei dem System wird ein zu den neusten Generationen von Benzin-, Diesel- und Kombiinstrument- Steuergeräten kompatibler Mikrocontroller von Motorola® verwendet. Die verwendeten ASIC's und Endstufen stammen ebenfalls aus den entsprechenden Steuergeräten. Somit ist für diese Anwendungsbereiche eine sehr hohe Identität mit dem Zielsystem vorhanden. In anderen Bereichen, wie z.B. der Fahrdynamik oder der Getriebesteuerung, sind die Anforderungen an die Mikrocontroller vergleichbar und an die I/O geringer. Aus diesem Grund werden dort häufig günstigere Mikrocontroller verwendet. Das hier vorgestellte System stellt für diese Bereiche daher eine Übermenge dar, was einen Einsatz ebenfalls ermöglicht.

- **Bauform vergleichbar mit Zielsystem**
  Die Abmessungen ohne Simulationsrechner sind nur unwesentlich größer als die des Zielsystems.

- **Umgebungsbedingungen vergleichbar mit Zielsystem**
  Das System ist für Umgebungsbedingungen (Temperatur, EMV, etc.) vergleichbar denen des Zielsystems ausgelegt.

- **Einfache Weiterverwendbarkeit der Ergebnisse**
  Erfahrungsgemäß beschränkt sich der Großteil der Softwareentwicklung auf die Anwendersoftware. Da die Anwendersoftware nur über definierte Schnittstellen auf Hardwaretreiber, Service Routinen sowie das Betriebssystem zugreift, können die erzielten Entwicklungsergebnisse einfach und ohne große Risiken im Seriensystem weiterverwendet werden.

  Die bei der FPGA Entwicklung erzielten Ergebnisse liegen als VHDL [Dou97] Beschreibungen vor. Diese VHDL Beschreibungen können als ausführbare Spezifikation z.B. für ASIC Entwicklungen verwendet werden.

Kapitel 8

AUSBLICK

8 Ausblick

8.1 Rekonfigurierbare Logik

Das im Rahmen dieser Arbeit dargestellte Entwicklungssystem unterstützt die Entwicklung und Erprobung moderner mikroelektronischer und mechatronischer Systeme durch eine sehr flexibel rekonfigurierbare Hardware. Begünstigt durch die in den letzten Jahren anhaltende Preisreduktion bei rekonfigurierbaren Logik- Bausteinen wird es zukünftig evtl. möglich sein, die mit dem Entwicklungssystem erzielten Ergebnisse direkt im Seriensystem einzusetzen. Damit würde der Umweg über eine ASIC-Entwicklung entfallen. Neben der Ersparnis der Einmalkosten könnte die Flexibilität gegenüber Änderungen der Anforderungen oder zur Fehlerbeseitigung bis weit über den Serienanlauf hinaus aufrecht-erhalten werden. Teilkomponenten, welche sich nicht durch rekonfigurierbare Bausteine abbilden lassen, könnten weiterhin als ASIC's realisiert werden. Anstelle der heute üblichen ASIC's, welche sowohl digitale, als auch analoge Komponenten beinhalten, könnte auf reine analoge ASIC's umgestellt werden. Damit wäre es möglich, eine für analoge Belange optimale Technologie einzusetzen.

Als weiterer Schritt wäre denkbar, nicht nur die im Rahmen dieser Arbeit betrachtete Signalauswertung und -generierung mit Hilfe von rekonfigurierbaren Bausteinen zu realisieren, sondern auch Teile oder den kompletten Mikrocontroller. Damit bei einer solchen Anwendung die rekonfigurierbare Logik optimal ausgenutzt werden kann wäre es beispielsweise denkbar, dass nicht immer die komplette Funktionalität gleichzeitig zur Verfügung stehen muss. Die Anwendung könnte in sinnvolle Anwendungsbereiche unterteilt und nur die für den entsprechenden Anwendungsbereich erforderlichen Funktionalitäten müssten dynamisch zur Verfügung gestellt werden.

Bei einem solchen Ansatz muss aber nicht zwangsläufig auf den Einsatz eines Mikrocontrollers verzichtet werden. Es könnten beispielsweise ein oder mehrere Mikrocontroller in der rekonfigurierbaren Logik verwendet werden (siehe RL-basierter Ansatz in Abbildung 8-1). Hierfür existieren bereits aktuell unterschiedliche Möglichkeiten. Zum einen können „soft cores" mit den durch die rekonfigurierbare Logik zur Verfügung gestellten Möglichkeiten realisiert werden. Zum anderen gibt es vermehrt auch rekonfigurierbare Logik Bausteine, welche bereits auf dem Silizium einen oder mehrere Mikrocontroller cores integriert haben. Bei diesen cores spricht man von „hard cores". Diese „hard cores" können ebenfalls sehr flexibel mit der rekonfigurierbaren Logik gekoppelt werden. Bei diesem Ansatz muss allerdings berücksichtigt werden, dass die häufig in Mikrocontrollern integrierten analogen Schaltungsteile wie z.B. A/D-Wandler meist nicht integriert sind und somit extern realisiert werden müssen.

Eine andere Möglichkeit besteht darin, Mikrocontroller mit rekonfigurierbarer Logik auszustatten (siehe MC-basierter Ansatz in Abbildung 8-1). Hierfür gibt es derzeit zwar noch keine direkt anwendbaren Beispiele. Aber einige Hersteller von Mikrocontrollern arbeiten bereits an Realisierungen.

Eine wichtige Voraussetzung für die erfolgreiche Umsetzung des oben beschriebenen Konzeptes besteht darin, Kriterien zu finden, welche die Zuordnung von Teilaufgaben auf den Mikrocontroller oder die rekonfigurierbare Logik unterstützt. Weiterhin gilt es anwendungsgerechte Beschreibungsformen zu finden, welche es ermöglicht, Teile relativ einfach zwischen dem Mikrocontroller und der rekonfigurierbaren Logik zu verschieben.

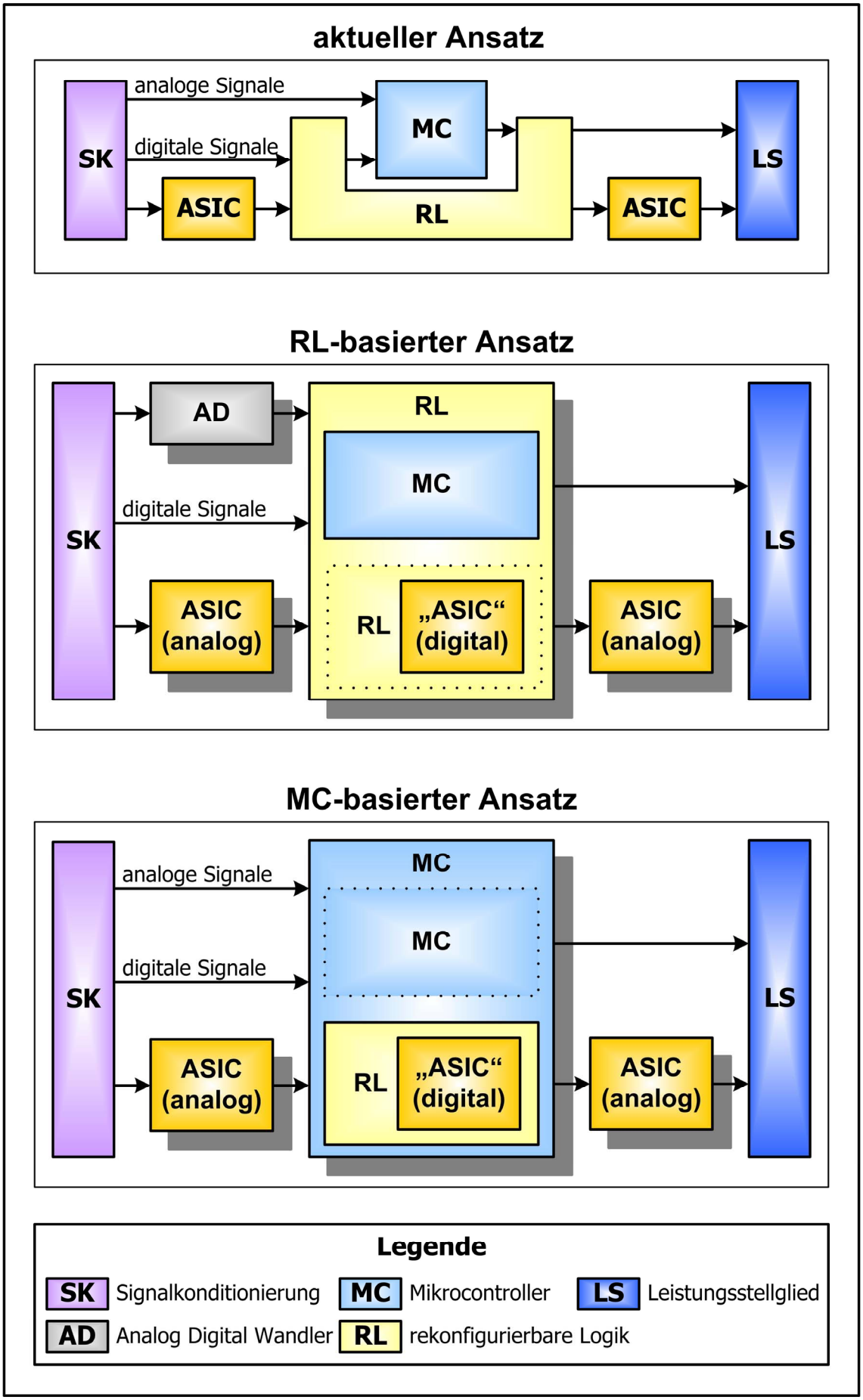


Abbildung 8-1: Rekonfigurierbare Logik in zukünftigen Seriensystemen

8.2 AUTOSAR

AUTOSAR ist eine Entwicklungspartnerschaft zwischen führenden Automobilherstellern, Zulieferern und Herstellern von Entwicklungswerkzeugen, die das Ziel hat einen industrieweiten Standard für die Elektrik/Elektronik-Architektur der Automobilindustrie zu definieren. Ausführlichere Informationen sind unter [AUTOSAR] zu finden.

Über die AUTOSAR Runtime Environment (RTE) wird beispielsweise eine einheitliche Schnittstelle zur Verfügung gestellt mit deren Hilfe Anwendersoftwaremodule unterschiedlicher Hersteller sehr einfach kombiniert und in unterschiedlichen Anwendungen verwendet werden können. (siehe Abbildung 8-2)

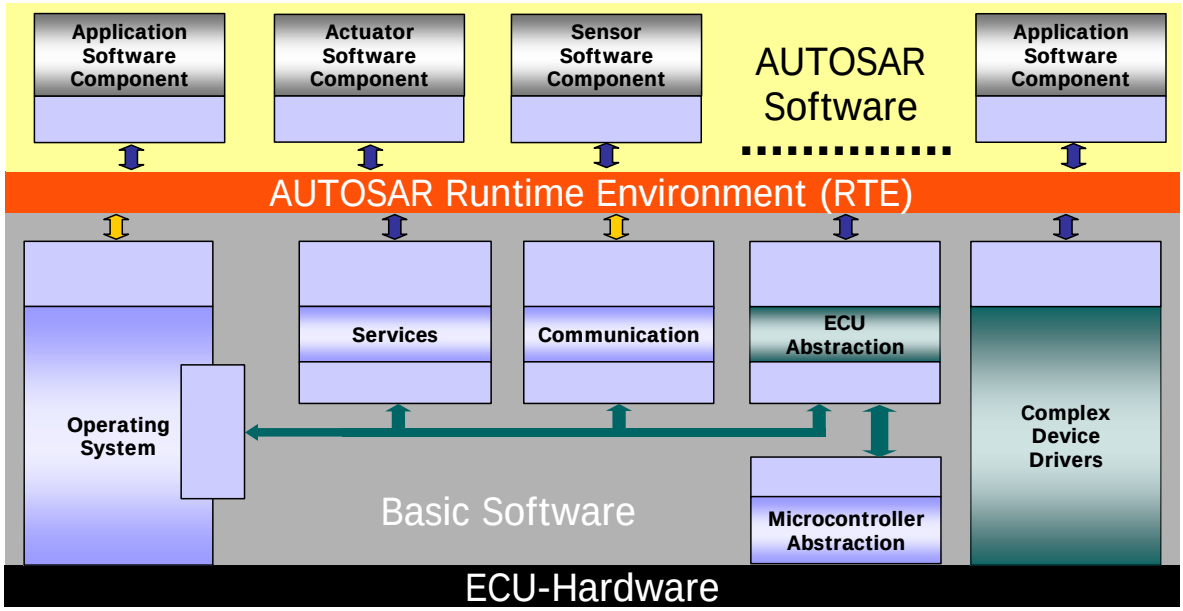


Abbildung 8-2: AUTOSAR Architektur

Sobald sich AUTOSAR als Standard etabliert hat und ausreichend erprobte AUTOSAR Komponenten verfügbar sind, kann die bisher eingesetzte CARTRONIC-Architektur durch die AUTOSAR-Architektur ersetzt werden. (siehe Abbildung 8-3)

Da zu erwarten ist, dass in den kommenden Jahren eine ständig steigende Anzahl an AUTOSAR Softwarekomponenten zur Verfügung steht, wird es einfach und schnell möglich sein, erprobte Softwarekomponenten beliebiger Hersteller zu verwenden.

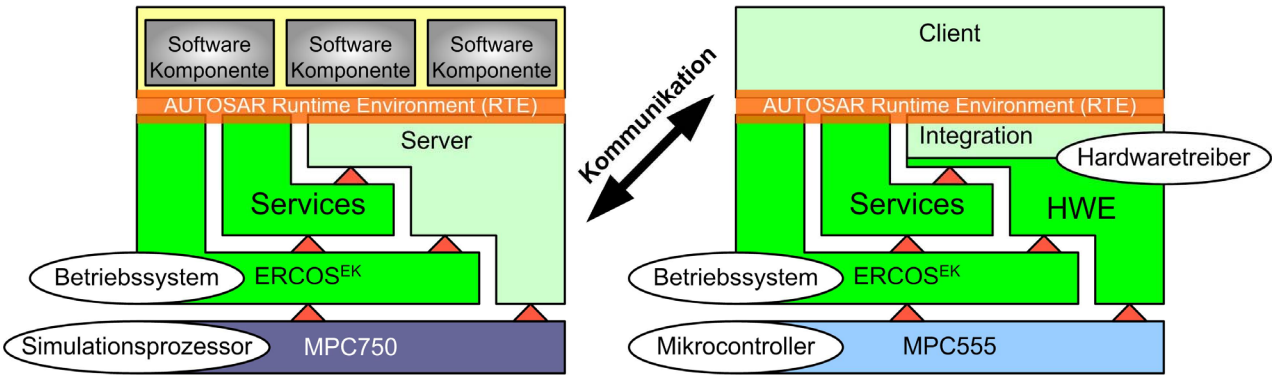


Abbildung 8-3: Architektur für zukünftige Entwicklungssystemen

Kapitel 9

THESEN

9 Thesen

- Durch die beim domänenspezifischen Rapid Prototyping in der Informations- und Elektrotechnik verwendete **Partionierung** von Serienstrukturen sowohl der Hardware, als auch der Software, in unabhängige Teilkomponenten sowie der Einführung entsprechender Schnittstellen wird die **Wiederverwendung** von Erfahrungen und Ergebnissen aus Serienentwicklungen systematisch ermöglicht.
- Durch diese systematische und konsequente **Wiederverwendung** wird es ermöglicht, die bislang den Systemlieferanten vorbehaltenen Erfahrungen und Ergebnissen aus Serienentwicklungen gut handhabbar den OEM sowie Spezialfirmen für Forschungs- und Entwicklungsaufgaben zur Verfügung zu stellen, ohne dass die Systemlieferanten ihr Know-How offen legen müssen. Das domänenspezifische Rapid Prototyping in der Informations- und Elektrotechnik ermöglicht damit effiziente Möglichkeiten für das **Simultaneous Engineering** zwischen OEM, Systemlieferant und Spezialfirma. Die OEM werden damit in die Lage versetzt, Innovationen selbst im eigenen Unternehmen voranzutreiben und somit früher als ihre Wettbewerber mit neuen Ideen in den Markt zu gehen.
- Mit Hilfe der **Wiederverwendung** von Erfahrungen und Ergebnissen aus Serienentwicklungen kann sehr schnell ein in wesentlichen Punkten mit dem **Zielsystem identischer Prototyp** erstellt und in der realen Umgebung eingesetzt werden. Der Entwicklungsaufwand kann damit auf neue Konzepte und Komponenten begrenzt werden und somit eine frühe Aussage über Realisierbarkeit und Akzeptanz getroffen werden. Dadurch können Doppel- und Parallelentwicklungen auf ein Minimum reduziert und damit die Entwicklungszeit gegenüber traditionellen Verfahren deutlich verkürzt werden.
- Durch den Einsatz **rekonfigurierbarer Logik** wird es ermöglicht, sehr einfach und schnell Änderungen und Erweiterungen in der normalerweise fixierten Hardware vorzunehmen. Dadurch wird es auch ermöglicht, projektspezifische Lösungen darzustellen.
- Durch Einbetten des Mikrocontrollers in die **rekonfigurierbare Logik** kann die normalerweise sehr früh zu treffende Entscheidung, wo eine Funktion realisiert werden soll (im Mikrocontroller, in der rekonfigurierbaren Logik oder aufgeteilt auf beide Einheiten), noch in sehr fortgeschrittenen Entwicklungsabschnitten beeinflusst werden.
- Durch den Einsatz **rekonfigurierbarer Logik** zur Entwicklung und Erprobung moderner mikroelektronischer und mechatronischer Systeme können heute übliche ASIC's, welche sowohl digitale, als auch analoge Komponenten beinhalten, als reine analoge ASIC's ausgeführt werden. Damit wird es ermöglicht, eine für analoge Belange optimale Technologie einzusetzen.
- Begünstigt durch die in den letzten Jahren anhaltende Preisreduktion bei rekonfigurierbaren **Logik- Bausteinen** wird es zukünftig möglich sein, diese direkt im **Seriensystem** einzusetzen.
- Die bei der Entwicklung der Anwendersoftware erzielten Ergebnisse können einfach und ohne großen Aufwand im Seriensystem **weiterverwendet** werden, da die Anwendersoftware nur über mit dem Zielsystem identische Schnittstellen auf Hardwaretreiber, Service Routinen sowie das Betriebssystem zugreift.
- Die in der rekonfigurierbaren Logik umgesetzten Funktionen können als **ausführbare Spezifikation** z.B. für ASIC Entwicklungen verwendet werden, da sie als VHDL Beschreibungen vorliegen.
- Das traditionelle **Rapid Prototyping** eignet sich vor allem für Grundsatz-untersuchungen in der frühen Entwicklungsphase ohne direkten Bezug zur Serie.

- Der **interne Bypass** ist am besten geeignet für Systeme, bei denen in der Regel kleinere Änderungen an der Software vorgenommen werden müssen.
- Der **externe Bypass** kann optimal für Systeme verwendet werden., bei denen die größten Teile der Hardware und der Software bereits fixiert sind
- Das **zielsystemidentische Rapid Prototyping** bringt die größten Vorteile für Systeme, bei denen zu erwarten ist, dass in der Hardware oder Software größere Modifikationen vorgenommen werden müssen.
- Durch die Steigerung des Automatisierungsgrades von Entwicklungsprozessen z.B. mit Hilfe der durchgängigen Verwendung von **grafischen Entwicklungswerkzeugen** mit automatischer **Codegenerierung** für Prototypen- und Seriensysteme, können Fehlerquellen reduziert und die Entwicklung beschleunigt werden.

Kapitel 10

ANHANG

10 Anhang

10.1 Hardware Encapsulation (HWE)

Die Hardware Encapsulation (HWE) stellt die Verbindung zwischen Anwendersoftware (physikalisches Modell) und Rechnerhardware dar. Sie hat die Aufgabe, die umfangreiche Funktionalität und damit einhergehend große Komplexität der Hardware auf eine schlanke Schnittstelle abzubilden. Durch diese Schnittstelle wird eine Abstraktionsebene eingeführt, die im Wesentlichen zwei Anforderungen genügt:

- Austausch der zugrunde liegenden Hardware ohne Änderungen in **den Anwender-prozessen.**
  Bei einem Wechsel der Rechnerhardware begrenzen sich die notwendigen Anpassungen ausschließlich auf die Hardwarekapselung. Der betroffene Bereich bleibt selbst dort durch die Gliederung in getrennte Ebenen eingegrenzt.
- **Einfacher Zugriff auf die Hardware.**
  Die Erstellung der Anwenderprozesse wird von den Interna der Hardware entkoppelt. Über Funktionsaufrufe, die den jeweiligen Belangen angepasst sind, wird von der Anwendersoftware auf die Hardware zugegriffen.

Neben diesen Anforderungen werden folgende weitere Leitgedanken verfolgt:

- Übergabe physikalischer Werte an der Schnittstelle zur Anwendersoftware.
- Einfacher Aufbau eines Projektes durch Hardwareanpassung ohne Eingriff in den Source-Code.
  Bei der Definition eines Projektes werden Festlegungen wie Zuordnung von Signalen zu Pins, Definition der Signalquelle/Senke, (A/D-Wandler, CAN, PWM...), Festlegung der zeitlichen Eigenschaften usw. getroffen. Diese Festlegungen beeinflussen die Berechnungen, die in der Hardwarekapselung ablaufen. Die Hardwarekapselung erlaubt diese Konfiguration durch Editieren von einheitlichen und klar strukturierten Ressourcen-Beschreibungstabellen. Eine Änderung des Codes ist nicht notwendig.
- Konsequente Nutzung des modularen Aufbaus der Hardware.
  Der Mikrocontroller besteht aus dem Rechnerkern, Speicher und intelligenten Peripheriemodulen. Die Peripheriemodule können autonom und zeitlich parallel zur CPU ihre Teilaufgaben erledigen. Die Interruptlast des Systems wird durch die Intelligenz und Zwischenspeichermöglichkeiten der Peripheriemodule deutlich reduziert.
- Passiver Aufbau der Hardwarekapselung.
  Die Hardwarekapselung ist im Sinne eines Schichtenmodells als Layer realisiert, der ausschließlich Informationen oder Dienste für die darüber liegenden Anwendersoftware zur Verfügung stellt. Sie nutzt keine Dienste der Anwenderprozesse. Sie stößt keinerlei Aktionen bei den Anwenderprozessen an. Die einzige Möglichkeit, welche die Hardwarekapselung hat, um sich der Anwenderprozesse mitzuteilen, ist ein Statuswort, welches als Teil der Schnittstellen-Information bereitgestellt wird.

10.1.1 Leistungsumfang der Hardwarekapsel

Die Hardwarekapselung soll ohne substantielle Änderungen für möglichst viele Projekte geeignet sein. Ein Ansatz, um dieser Anforderung zu genügen ist es, nur elementare Zugriffsmechanismen auf die Hardware bereitzustellen. Damit wäre ein großer gemeinsamer Nenner zu finden, allerdings mit dem Preis, die Schnittstelle sehr komplex gestalten zu müssen.

Der genau entgegen gesetzte Ansatz wäre der, alle hardwarenahe Funktionalität, die sich in der Vereinigungsmenge aller Projekte finden lässt, zu realisieren und zu versuchen, sie auf die schlanke Schnittstelle abzubilden. Abgesehen von den zweifelhaften Erfolgsaussichten wäre dieser Ansatz mit ständigen Änderungen der Hardwarekapselung verbunden.

Der hier verfolgte Ansatz ist in der Mitte angesiedelt. Die Hardwarekapselung leistet über elementare Zugriffsfunktionalität hinaus auch Aufgaben der Koordination und der Signalumsetzung.

Die Aufgaben der Hardwarekapselung werden in 2 Teilaufgaben unterteilt:

1. Bedienung der Ein- und Ausgänge
- Analogwerterfassung
- Digitalwerterfassung und -ausgabe
- PWM-Signalwerterfassung und -ausgabe
- CAN-Kommunikation
- Bedienung der seriellen Schnittstellen

2. Rechnerkerninterne Routinen
- EEPROM-Bearbeitung
- Flashprogrammierung
- Rechner-Selbsttest

10.1.2 Architektur der Hardwarekapsel

Die Kommunikation zwischen Anwenderprozess und Hardwarekapselung erfolgt über die Komponententreiber, die Bestandteil der Anwenderprozesse sind.

Die Architektur der Hardwarekapselung besteht aus 3 Schichten:

- Der Hardwarezugriffsschicht
- Der Umsetzungsschicht
- Dem Hardwaretreiberkoordinator.

Der **Hardwaretreiberkoordinator** hat interne und externe Koordinationsaufgaben. Extern entscheidet er, ob über einen Eingriff von außen gesteuert (Remote-Control, Bypass) die Größe umgeleitet werden soll. Als interne Aufgaben steuert der Koordinator die Umsetzungs- und Hardwarezugriffsschicht. Hierzu werden konfigurierbare Tabellen abgearbeitet und bestimmte Routinen aufgerufen. Diese Routinen sind signalunabhängig formuliert.

Die **Umsetzungsschicht** stellt Dienste auf physikalischer Ebene zur Verfügung. Sie berechnet bei den Eingangsgrößen, aus den Rohwerten die physikalischen Werte und bei den Ausgangsgrößen aus den physikalischen Werten die entsprechenden Rechnerwerte, welche die Hardwarezugriffsschicht für die Ansteuerung der Rechnerhardware benötigt. Die Adressen der notwendigen Parameter, die für die Umsetzung benötigt werden, sind in Tabellen abgelegt.

Die **Hardwarezugriffsschicht** greift direkt auf die µC-Register zu und bietet dem Hardwaretreiberkoordinator Dienste an. Neben den Hauptaufgaben (Erfassungs-,

Ausgabe-, Flashprogrammierroutinen,...) gibt es Funktionen für die Ansteuerung der jeweiligen Rechnerhardware zur Initialisierung und Überwachung des jeweiligen Peripheriebausteins.

10.1.3 Konfigurationsoberfläche für die Hardwarekapsel

Um die große Menge an Konfigurationsdaten übersichtlich einstellen und verwalten zu können, wird ein spezielles Konfigurationswerkzeug zur Verfügung gestellt. Im folgenden Bild kann ein kleiner Ausschnitt der ADC Konfiguration betrachtet werden. Mit Hilfe von SCON werden die in übersichtlichen Oberflächen eingestellten Konfigurationsdaten anschließend in „C“ und Header Files abgelegt. Diese können in Projekte eingebunden werden.

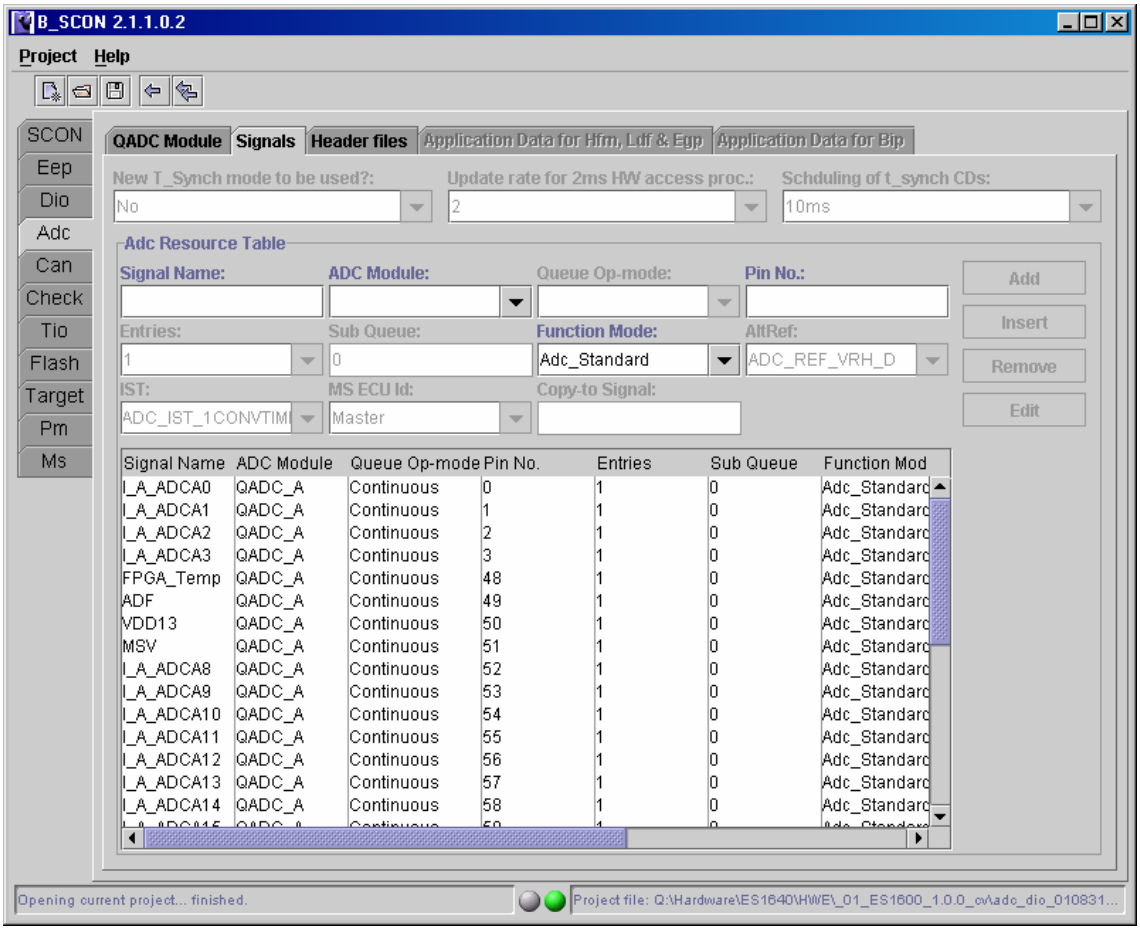


Abbildung 10-1: SCON Konfigurationswerkzeug

10.1.4 Integration der HWE am Beispiel der Analogwerterfassung

Die Abbildung 10-2 gibt einen Einblick in den Aufruf der HWE für die Klasse I_A_ADCA0. Mit der Methode „do_IO“ werden sowohl der Status und der Registerwert ausgelesen, als auch mit Hilfe einer Interpolationsroutine der Services der physikalische Wert berechnet.

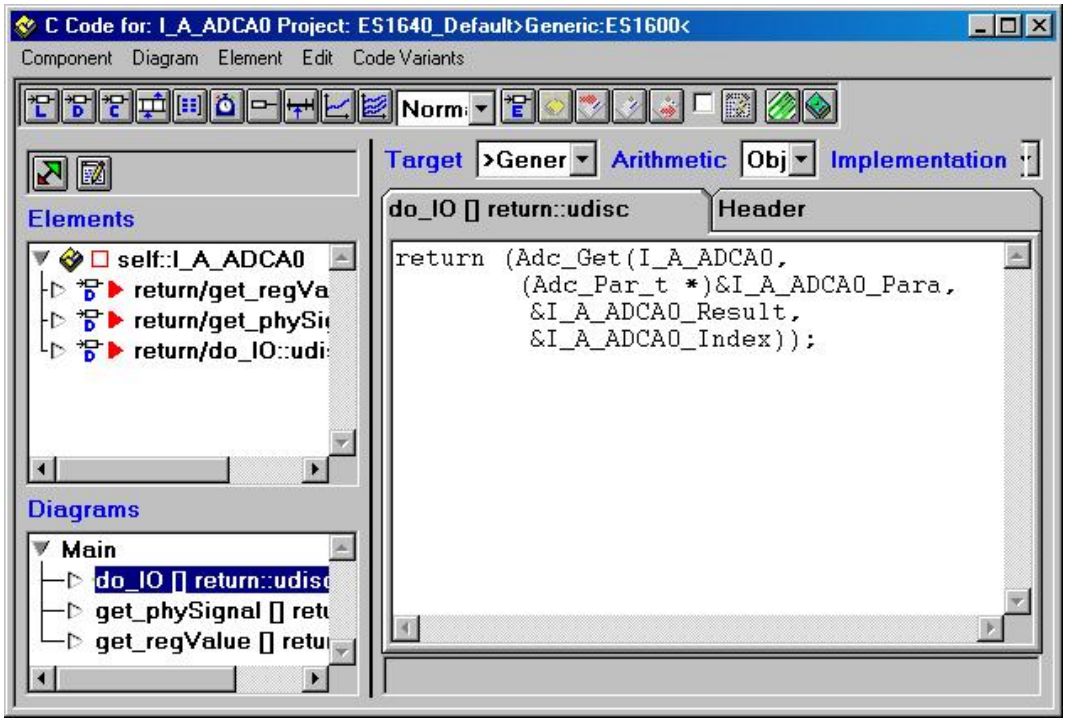


Abbildung 10-2: ADC Methodenaufruf do_IO

Die Methode „get_phySignal“ gibt den physikalischen Wert zurück.

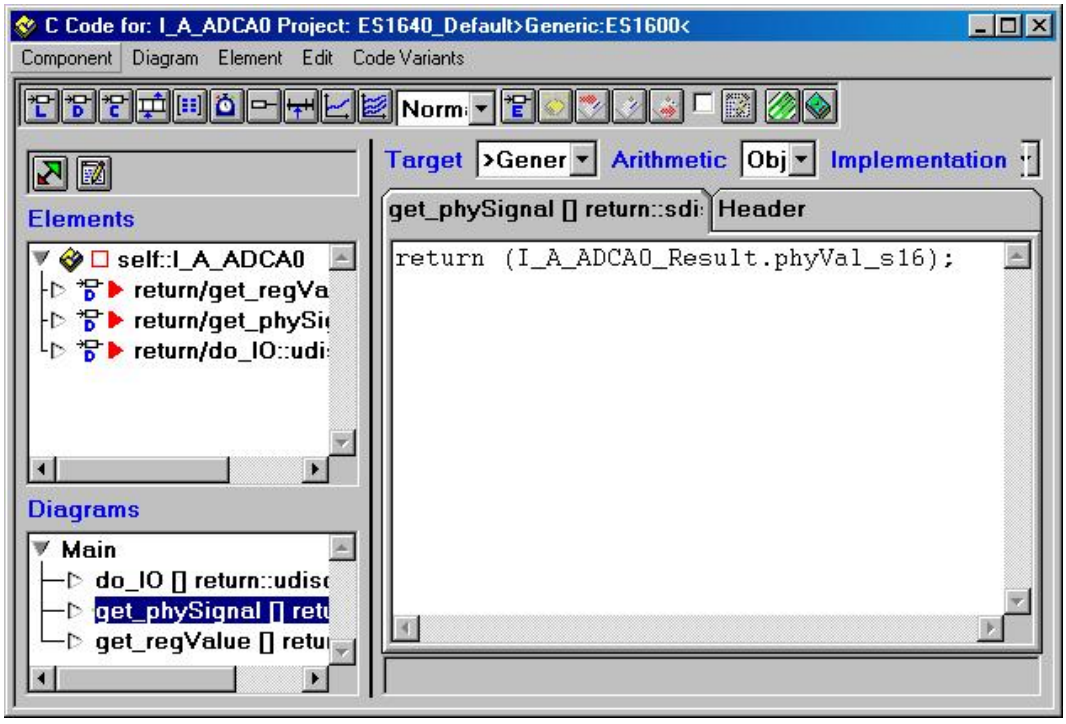


Abbildung 10-3: ADC Methodenaufruf get_phySignal

Die Methode „get_regValue“ gibt den Registerwert des ADC zurück.

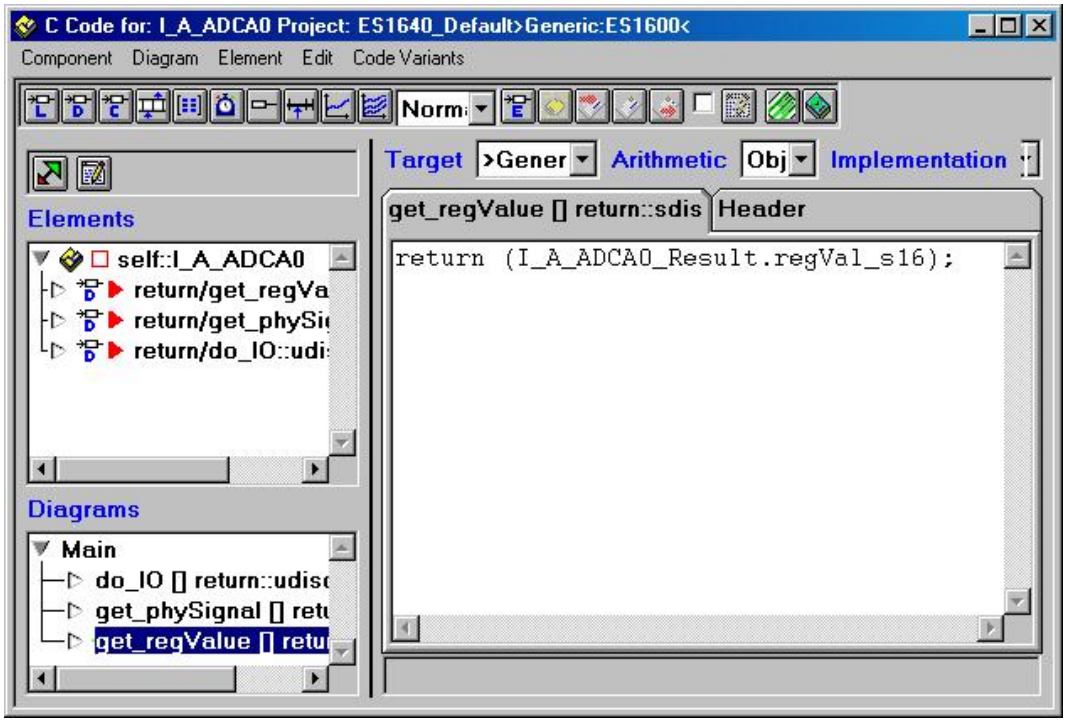


Abbildung 10-4: ADC Methodenaufruf get_regValue

Im Header, welcher in Abbildung 10-5 zu sehen ist, werden Informationen wie unterer Grenzwert (SRCLow), oberer Grenzwert (SRCHigh) sowie die Stützstellen für die Interpolation definiert. Der untere Grenzwert bewirkt, dass beim Unterschreiten dieses Wertes eine entsprechende Statusmeldung zurückgegeben wird. Analog dazu verhält sich der obere Grenzwert beim Überschreiten.

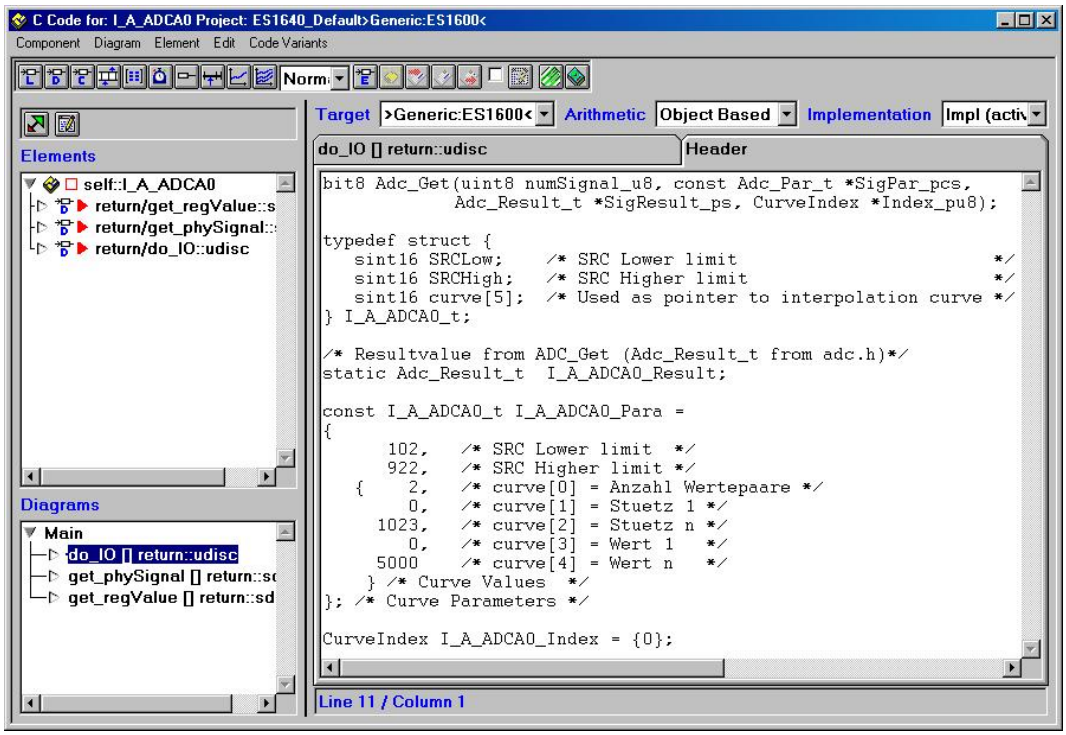


Abbildung 10-5: ADC Header

10.2 Service Routinen

Wie bei den Hardwaretreibern werden auch die Services aus aktuellen Serienprojekten von Diesel- bzw. Benzinmotorsteuergeräten der Robert BOSCH GmbH verwendet. Diese Services wurden von den Geschäftsbereichen für Dieselsysteme (DS) und Benzinsysteme (GS) entwickelt und im Rahmen dieser Arbeit eingebunden. In Tabelle 10-1 sind die von den Services zur Verfügung gestellten Routinen dargestellt.

Name	Beschreibung
Min	gibt die kleinere der beiden Zahlen zurück
Max	gibt die größere der beiden Zahlen zurück
Limit	gibt den durch min und max begrenzten Wert von (x) zurück
Abs32	gibt den Betrag von (x) zurück
Abs16	gibt den Betrag von (x) zurück und begrenzt das Ergebnis auf [0, MAXSINT16]
AddU32	berechnet (x + y) und begrenzt das Ergebnis auf [0, MAXUINT32]
AddS32	berechnet (x + y) und begrenzt das Ergebnis auf [MINSINT32, MAXSINT32]
AddU16	berechnet (x + y) und begrenzt das Ergebnis auf [0, MAXUINT16]
AddS16	berechnet (x + y) und begrenzt das Ergebnis auf [MINSINT16, MAXSINT16]
SubU32	berechnet (x - y) und begrenzt das Ergebnis auf [0, MAXUINT32]
SubS32	berechnet (x - y) und begrenzt das Ergebnis auf [MINSINT32, MAXSINT32]
SubU16	berechnet (x - y) und begrenzt das Ergebnis auf [0, MAXUINT16]
SubS16	berechnet (x - y) und begrenzt das Ergebnis auf [MINSINT16, MAXSINT16]
MulU32	berechnet (x * y) und begrenzt das Ergebnis auf [0, MAXUINT32]
MulS32	berechnet (x * y) und begrenzt das Ergebnis auf [MINSINT32, MAXSINT32]
MulU16	berechnet (x * y) und begrenzt das Ergebnis auf [0, MAXUINT16]
MulS16	berechnet (x * y) und begrenzt das Ergebnis auf [MINSINT16, MAXSINT16]
MulShiftU32	berechnet ((x * y) >> n) und begrenzt das Ergebnis auf [0, MAXUINT32]
MulShiftS32	berechnet ((x * y) >> n) und begrenzt das Ergebnis auf [MINSINT32, MAXSINT32]
MulShiftU16	berechnet ((x * y) >> n) und begrenzt das Ergebnis auf [0, MAXUINT16]
MulShiftS16	berechnet ((x * y) >> n) und begrenzt das Ergebnis auf [MINSINT16, MAXSINT16]
MulDivU16	berechnet ((x * y) / z) und begrenzt das Ergebnis auf [0, MAXUINT16]
MulDivS16	berechnet ((x * y) / z) und begrenzt das Ergebnis auf [MINSINT16, MAXSINT16]
DivU32	berechnet (x / y) und begrenzt das Ergebnis auf [0, MAXUINT32]
DivS32	berechnet (x / y) und begrenzt das Ergebnis auf [MINSINT32, MAXSINT32]
DivU16	berechnet (x / y) und begrenzt das Ergebnis auf [0, MAXUINT16]
DivS16	berechnet (x / y) und begrenzt das Ergebnis auf [MINSINT16, MAXSINT16]
ModU32	berechnet Rest(x / y) und begrenzt das Ergebnis auf [0, MAXUINT32]
ModS32	berechnet Rest(x / y) und begrenzt das Ergebnis auf [MINSINT32, MAXSINT32]
Random	gibt eine Zufallszahl zurück
AvrgS16	gibt den Mittelwert der beiden Zahlen zurück
AvrgU16	gibt den Mittelwert der beiden Zahlen zurück
P	Proportionalglied
I	Integrationsglied
PI	PI-Glied
PT1	Proportionalglied mit Verzögerung
DT1	Differenzierglied mit Verzögerung
DT1Win	Differenzierglied mit Verzögerung und Steigungsabhängigkeit
PDT1	Proportional- und Differenzierglied mit Verzögerung
PDT1Win	Proportional- und Differenzierglied mit Verzögerung und Steigungsabhängigkeit
IpoCurve	Kennlinieninterpolation
IpoMap	Kennfeldinterpolation
IpoSearchInterval	Sucht Interpolationsintervall für Gruppenkennlinien und -Felder

IpoGroupCurve	Gruppenkennlinieninterpolation
IpoGroupMap	Gruppenkennfeldinterpolation
Ramp	Rampenfunktion
RampDir	Rampenfunktion mit Richtungsvorgabe
RampSwitch	Umschalter mit rampenförmiger Unterdrückung der Umschaltsprünge
Hysteresis	Hysteresekennlinie
Debounce	Entprellt digitale Eingänge
TransStage	Hilfsfunktion für Sensorersatzwerte
MemCopy	Kopiert Speicherbereiche
MemFill	Füllt den Speicher
SetBit8	Setzt ein Bit in einer uint8 Variable
SetBit16	Setzt ein Bit in einer uint16 Variable
SetBit32	Setzt ein Bit in einer uint32 Variable
GetBit8	Liest ein Bit aus einer uint8 Variable
GetBit16	Liest ein Bit aus einer uint16 Variable
GetBit32	Liest ein Bit aus einer uint32 Variable
SetBitField8	Setzt mehrere Bits in einer uint8 Variable
SetBitField16	Setzt mehrere Bits in einer uint16 Variable
SetBitField32	Setzt mehrere Bits in einer uint32 Variable
GetBitField8	Liest mehrere Bits aus einer uint8 Variable
GetBitField16	Liest mehrere Bits aus einer uint16 Variable
GetBitField32	Liest mehrere Bits aus einer uint32 Variable
Accumulator	Summiert die Eingangswerte auf
Counter	Zähler der bei jedem Aufruf um 1 inkrementiert wird
ClosedInterval	Komparator: min <= x <= max
LeftOpenInterval	Komparator: min < x <= max
OpenInterval	Komparator: min < x < max
RightOpenInterval	Komparator: min <= x < max
GreaterZero	Komparator: x > 0

Tabelle 10-1 Services

10.3 Geberräder und Sensoren zur Drehzahl- und Positionserfassung

10.3.1 Geberräder von Kurbel- und Nockenwelle

Die beiden Geberräder sind an der Kurbel- bzw. der Nockenwelle angebracht. Das Geberrad der Kurbelwelle wird auch als Inkrementrad und das der Nockenwelle als Phasenrad bezeichnet. Weiterhin ist zu beachten, dass sich die Kurbelwelle und somit auch das Inkrementrad mit der doppelten Geschwindigkeit wie die Nockenwelle dreht. Ein vollständiges Arbeitsspiel einer Verbrennungskraftmaschine erstreckt sich über 720° Kurbelwellen-Winkel (kurz: 720° KW).

10.3.1.1 Kurbelwellengeberräder (Inkrementgeberräder)

In Europa werden üblicherweise Kurbelwellengeberräder (Inkrementräder) mit 60 Zähnen verwendet. Es gibt aber auch Räder mit anderen Zähnezahlen, wie z.B. mit 36 Zähnen von Toyota. Die folgenden Betrachtungen beschränken sich auf Inkrementräder mit 60 Zähnen. Das entspricht einer Winkelteilung von Zahn zu Zahn von genau 6°. Üblicherweise werden die fallenden Flanken der Inkrementräder ausgewertet.

Weiter unterscheiden sich die Inkrementräder in der Anzahl der fehlenden Zähne (Lücken). Die Lücken kennzeichnen die so genannten Bezugsmarken. Es fehlen normalerweise 1, 2 oder 3 Gruppen zu jeweils 2 Zähnen. Man spricht dann von einem 60-2, 60-2x2 oder 60-3x2 Inkrementrad. Die folgende Tabelle gibt an, für welche Anwendungsfälle welche Inkrementräder im Allgemeinen verwendet werden:

Kurbelwelle (Crank - Shaft)						
Anz. Zylinder	3	4	5	6	8	10
Anz. Zähne	60	60	60	60	60	60
Zahnteilung	6°	6°	6°	6°	6°	6°
Anz. Lücken	3	2	1	1	1	1
Lückenbreite	2 Zähne	2 Zähne	2 Zähne	2 Zähne	2 Zähne	2 Zähne

Tabelle 10.2: Zähne Kurbelwelle

Den folgenden Bildern kann die Anordnung der Zähne und Lücken entnommen werden.

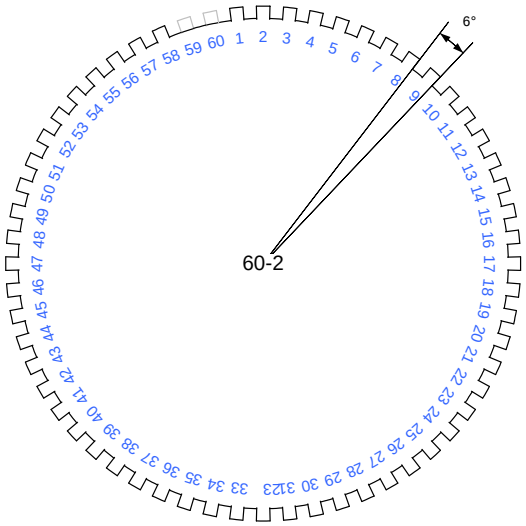


Abbildung 10-6: Geberrad 60-2

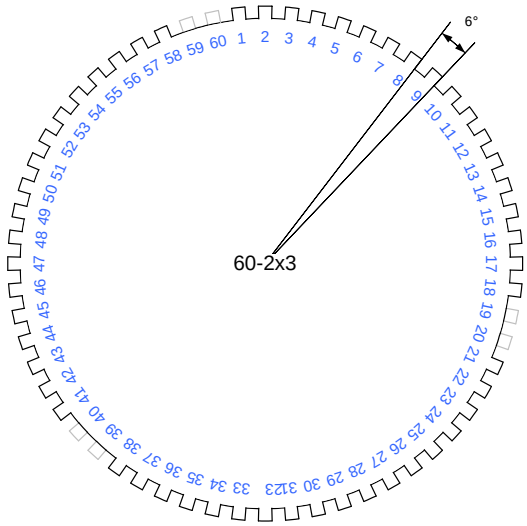


Abbildung 10-7: Geberrad 60-2x3

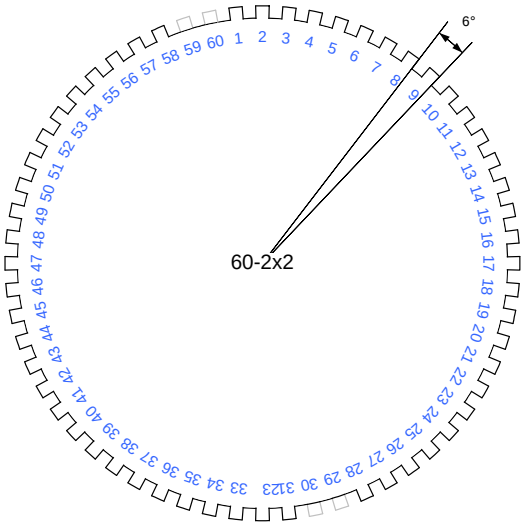


Abbildung 10-8: Geberrad 60-2x2

10.3.1.2 Nockenwelleengeberräder (Phasenräder)

Im einfachsten Fall besitzt das Geberrad der Nockenwelle (Phasenrad) nur einen einzigen Zahn. Dieser ist meist so ausgerichtet, dass er während der Lücke des Inkrementrades einmal ein Signal und beim nächsten Mal kein Signal (bedingt durch die halbe Drehzahl des Phasenrades) liefert. Das ermöglicht eine eindeutige Zuordnung des KW- Bereiches, in welchem sich das System befindet, nämlich 0°- 360° KW oder 360° - 720° KW. Solche einfachen Phasenräder werden fast nicht mehr verwendet, da die Positionserkennung mindestens 360° KW dauert. Man verwendet deshalb meist Phasenräder mit mehreren Zähnen, die so genannten Schnellstart-Phasenräder. Ein Schnellstart-Phasenrad hat gegenüber dem Standard-Phasenrad aber noch einen weiteren Vorteil. Fällt beispielsweise der Inkrementsensor aus, so ist es möglich, den Motor im „Notlauf“ zu betreiben. Die Auflösung des Phasensignals ist jedoch zu ungenau, um den Inkrementsensor auch im Normalbetrieb zu ersetzen.

In der folgenden Tabelle ist eine Aufstellung, für welche Anwendungsfälle welche Phasenräder im Allgemeinen verwendet werden.

Nockenwelle (Cam - Shaft)						
Anz. Zylinder	3	4	5	6	8	10
Anz. Zähne	3 + 2	4 + 3	5 + 2	5 + 2	5 + 2	5 + 2

Tabelle 10.3: Zähne Nockenwelle

Es gibt sehr viele verschiedene Varianten von Phasenrädern. Im Folgenden sind einige beispielhaft dargestellt:

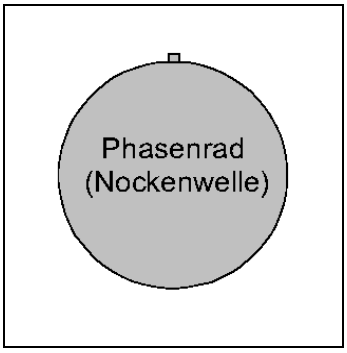


Abbildung 10-9: Standard Phasenrad

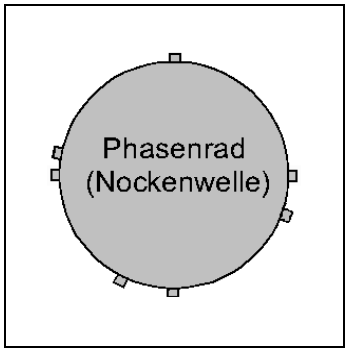


Abbildung 10-10: Schnellstart Phasenrad (VW)

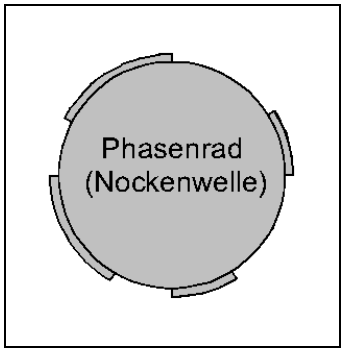


Abbildung 10-11: Schnellstart Phasenrad (BOSCH)

10.3.2 Sensoren von Kurbel- und Nockenwelle

In der Regel werden derzeit für die Signalerfassung der Kurbelwelle noch induktive Sensoren verwendet. Neuerdings versucht man auch in den Simmerring integrierte magnetoresistive Sensoren (MR-Sensoren) einzusetzen.

Aufgrund der relativ geringen Durchmesser von Nockenwellen-Geberrädern bei PKW wird im Gegensatz zu NKW gewöhnlich ein Hall-Sensor anstelle eines induktiven Sensors verwendet. Im Folgenden werden verschiedene Sensoren beschrieben.

10.3.2.1 Induktive Sensoren

Induktive Sensoren bestehen aus einem Dauermagneten sowie einem Weicheisenkern mit einer Kupferwicklung, im Folgenden Spule genannt. Der Weicheisenkern schließt an den Dauermagneten an. Das Magnetfeld des Dauermagneten erstreckt sich über den Weicheisenkern (Polstift) bis in das ferromagnetische Geberrad hinein. Der magnetische Fluss durch die Spule hängt nun davon ab, ob dem Polstift eine Lücke oder ein Zahn gegenübersteht. Ein Zahn bündelt den Streufluss der magnetischen Anordnung und es kommt zu einer Verstärkung des Flusses durch die Spule. Eine Lücke dagegen schwächt den Fluss ab. So wird in der Spule eine der zeitlichen Änderung des Magnetflusses proportionale Spannung induziert. Bei einer gleichmäßigen Zahnstruktur (ohne Lücken) ergibt sich annähernd ein sinusförmiger Spannungsverlauf. Die Drehzahl ergibt sich aus dem Abstand der Nulldurchgänge dieser Spannung. Die Amplitude hängt dabei sehr stark (exponentiell) vom Luftspalt ab. Die Amplitude ist zudem noch abhängig von der Zahngröße und der Geschwindigkeit des Geberrades. Zähne können bis zu Luftspaltbreiten von der Hälfte oder einem Drittel eines Zahnabstandes noch einwandfrei detektiert werden. Mit den üblichen Kurbelwellen-Geberrädern werden Luftspalte von 0,8 mm bis zu 1,5 mm abgedeckt. [Bos02_2]

Quelle: [Bos02_1]

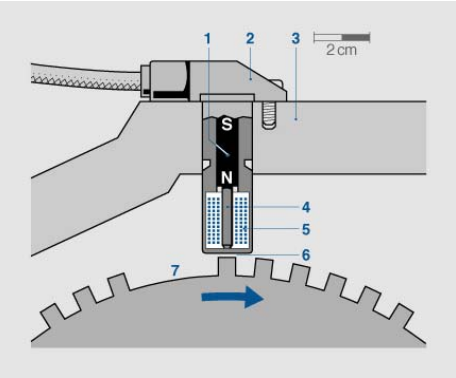


Abbildung 10-12: Aufbau Induktiver Drehzahlsensor

Quelle: [Bos02_1]

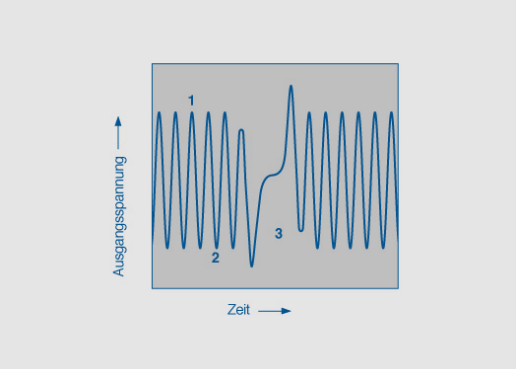


Abbildung 10-13: Signalverlauf Induktiver Drehzahlsensor

10.3.2.2 Hall-Effekt Sensoren

Diese Sensoren nutzen den Hall-Effekt, um einen Zahn bzw. eine Lücke zu detektieren.

Hall-Effekt

Legt man an einen Leiter eine Spannung und senkrecht dazu ein Magnetfeld an, entsteht senkrecht zur Stromrichtung und zum Magnetfeld eine Spannung (Hall-Spannung U_H)

Quelle: [Bos02_1]

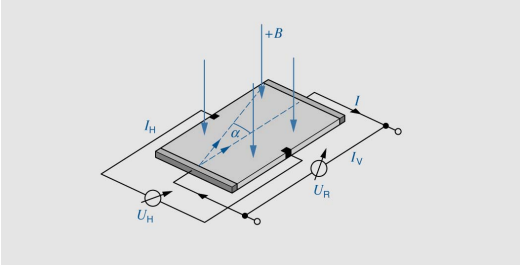


Abbildung 10-14: Hall-Effekt

U_H: Hall-Spannung
R: Hall-Konstante
I_V: Versorgungsstrom
B: magnetische Induktion
d: dicke des Hall-Elements

$$U_H = R \cdot I_V \cdot \frac{B}{d}$$

Gleichung 10.1 Hall-Spannung

Aufbau und Arbeitsweise

Bei einem Hall-Stabsensor befindet sich das Hall-Element zwischen dem ferromagnetischen Geberrad und einem Dauermagneten. Der Dauermagnet liefert ein Magnetfeld senkrecht zum Hall-Element. Passiert nun ein Zahn das stromdurchflossene Sensorelement des Stabsensors, so verändert er die Feldstärke des Magnetfeldes senkrecht zum Hall-Element. Dadurch entsteht ein Spannungssignal (Hall-Spannung), das im Millivolt-Bereich liegt und unabhängig von der Relativgeschwindigkeit zwischen dem Sensor und dem Geberrad ist. Eine im Sensor integrierte Auswerteelektronik bereitet das Signal auf und gibt es als Rechtecksignal aus.

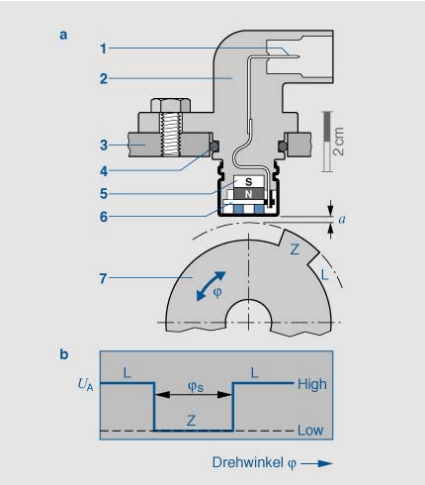


Abbildung 10-15: Hall-Stabsensor

Quelle: [Bos02_1]

1: Elektrischer Anschluss
2: Sensorgehäuse
3: Motorgehäuse
4: Dichtung
5: Dauermagnet
6: Hall-Element
7: Geberrad
a: Luftspalt

10.3.3 Signalverläufe unterschiedlicher Geberräder

In diesem Abschnitt werden zunächst die Messungen an einem realen Kurbelwellengeberrad mit 60-2 Zähnen bei unterschiedlichen Drehzahlen dargestellt. In einem weiteren Abschnitt werden die idealisierten Kurvenverläufe für Kurbelwellengeberräder mit 60-2 sowie 60-2x2 Zähnen dargestellt.

10.3.3.1 Realer Signalverlauf eines 60-2 Kurbelwellengeberrades

In den folgenden Bildern sind die Messungen realer Signalverläufe bei unterschiedlichen Drehzahlen dargestellt. Im jeweils rechten Bild sind 2 volle Umdrehungen der Kurbelwelle zu sehen. Im linken Bild ist der Bereich um die Lücke herausgezoomt. Den Kanälen am Oszilloskop sind folgende Signale zugeordnet:

- Sensorsignal des induktiven Kurbelwellensensors (1)
- Digitalisiertes Signal des Kurbelwellensensors (4)
- Simuliertes Open Kollektor Signal eines Nockenwellensensors (2)

Die Abhängigkeit der Amplitude von der Drehzahl kann in den Messungen ebenfalls gut beobachtet werden.

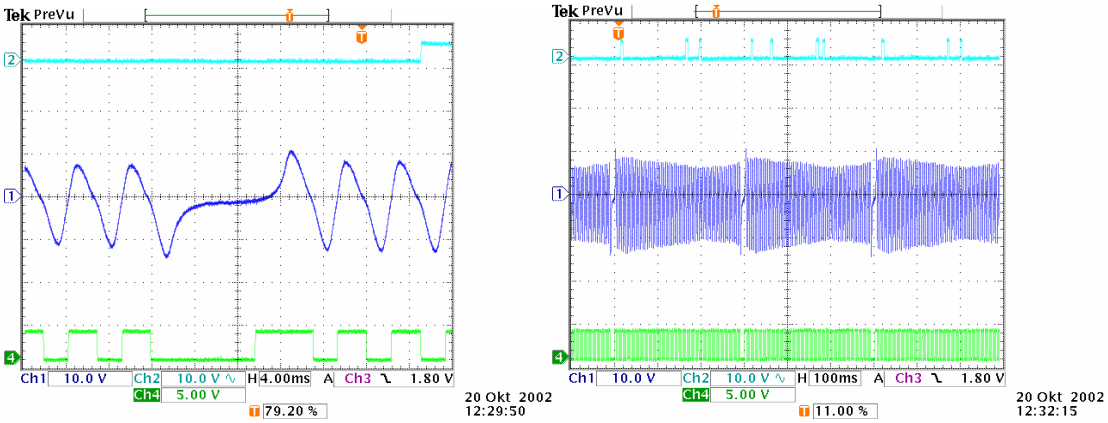


Abbildung 10-16: Reales Inkrementsignal 200 1/min

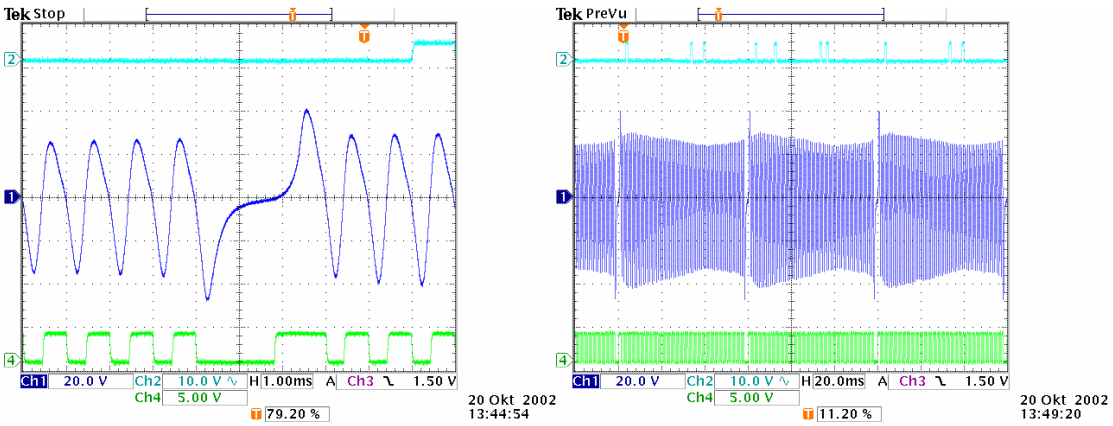


Abbildung 10-17: Reales Inkrementsignal 1000 1/min

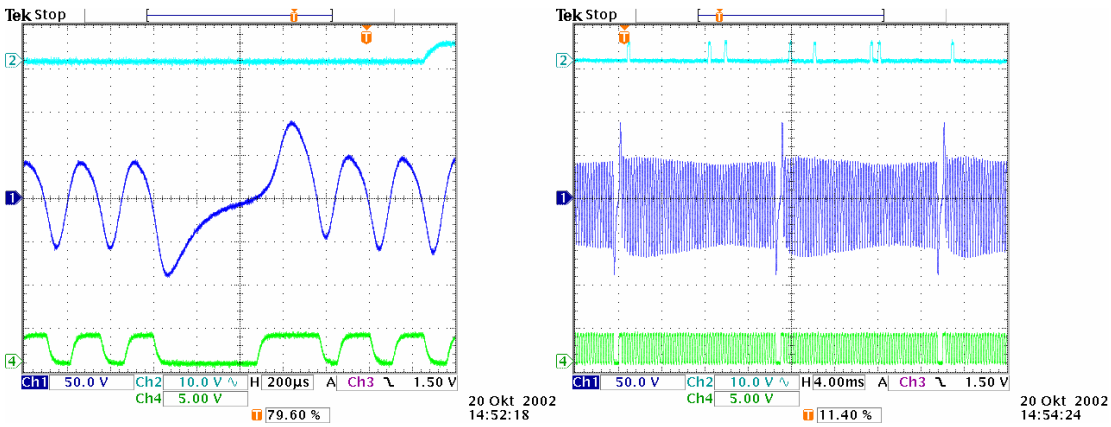


Abbildung 10-18: Reales Inkrementsignal 4000 1/min

10.3.3.2 Idealisierte Signalverläufe für unterschiedliche Geberräder

In den folgenden Bildern sind die idealisierten Signalverläufe eines induktiven Inkrementgebers sowie das daraus generierte digitale Signal dargestellt. Die rot gepunktet eingezeichneten Zähne dienen dabei nur der Verdeutlichung; im Realen sind diese nicht vorhanden. Zudem sind bei realen Signalen die Flanken des Zahns vor und nach der Lücke etwas verschoben. Messschriebe realer Kurvenverläufe sind im Kapitel 10.3.3.1 Realer Signalverlauf eines 60-2 zu finden.

Des Weiteren sind die Signale des nach dem Hall-Prinzip arbeitenden Phasengebers dargestellt. Der Sensor besitzt bereits intern eine elektronische Schaltung, um die Signale aufzubereiten und kann somit direkt einen digitalen Open-Kollektor-Ausgang zur Verfügung stellen.

10.3.3.3 Idealisierter Signalverlauf eines 60-2 Kurbelwellengeberrades

Bei diesem Inkrementgeberrad handelt es sich um ein 60-2 Geberrad, erkennbar an den 2 Lücken innerhalb 720° KW. Als 0° Bezug wird die zweite fallende Flanke nach der Lücke definiert. Als Segment bezeichnet man den Bereich, welcher beiderseits durch die zweite fallende Flanke nach einer Lücke begrenzt wird. Bei einem 60-2 Geberrad ergeben sich demzufolge zwei Segmente.

Das Phasengeberrad besitzt genau einen Zahn. Damit ist es möglich, die beiden 360° Segmente eindeutig zu identifizieren.

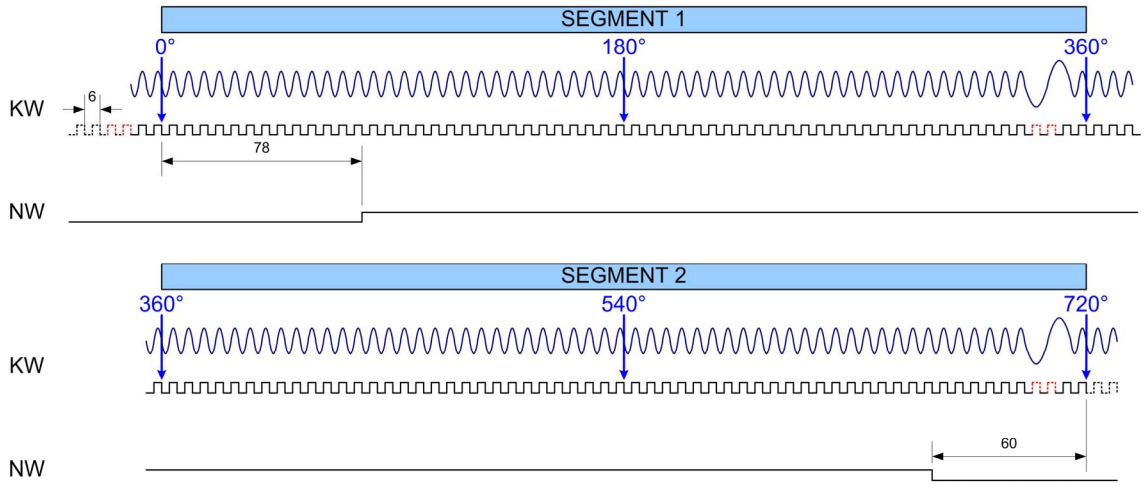


Abbildung 10-19: Idealisierter Signalverlauf 60-2 Inkrementgeberrad

10.3.3.4 Idealisierter Signalverlauf eines 60-2x2 Kurbelwellengeberrades

Bei diesem Inkrementgeberrad handelt es sich um ein 60-2x2 Geberrad, zu erkennen an den vier Lücken je 720° KW. Es ergeben sich demzufolge vier Segmente. Das Phasengeberrad besitzt genau sieben Zähne. Sie sind so angeordnet, dass jedes 180° Segment eindeutig identifiziert werden kann.

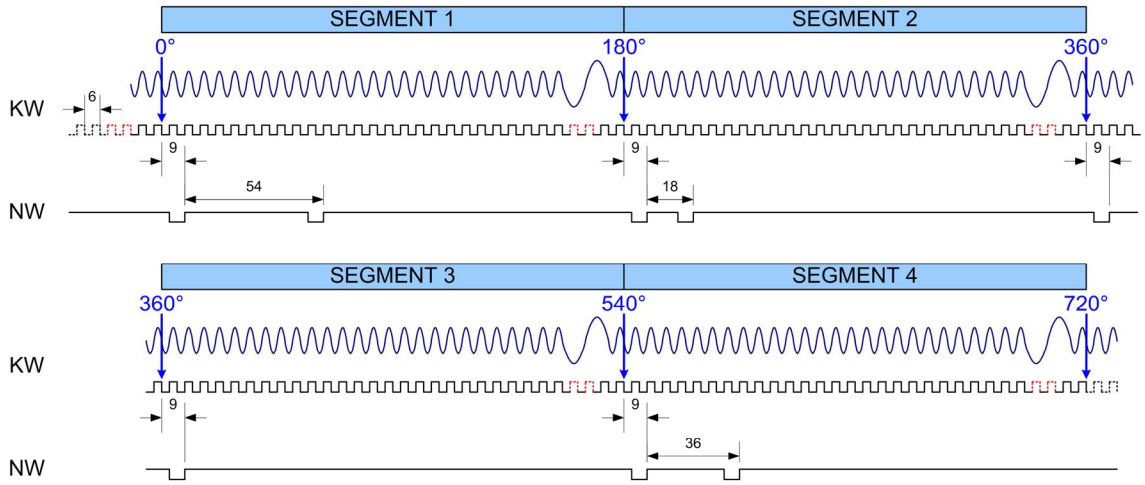


Abbildung 10-20: Idealisierter Signalverlauf 60-2x2 Inkrementgeberrad

ABKÜRZUNGSVERZEICHNIS

ABI
Adaptation **B**oard **I**nterface

ABS
Anti **B**lockier **S**ystem

ACC
Active **C**ruise **C**ontrol

A/D
Analog / **D**igital

AFS
Active **F**ront **S**teering

AML
ASAP **M**eta **L**anguage (Beschreibungssprache für Schnittstellendaten)

API
Application **P**rogram **I**nterface (Schnittstellenbeschreibung)

ASAM-MCD
Arbeitskreis zur **S**tandardisierung von **A**utomations- und **M**esssystemen, mit den Arbeitsgruppen **M**easurement, **C**alibration and **D**iagnostics

ASAM-MCD-2MC
Standard-Austauschformat zur Programmbeschreibung für Applikationszwecke.

ASIC
Application **S**pecific **I**ntegrated **C**ircuit

AUTOSAR
Automotive **O**pen **S**ystem **Ar**chitecture

BDE
Block**d**iagramm **E**ditor

BDM
Background **D**ebug **M**ode

BIP
Begin of **I**njection **P**oint

CAN
Controller **A**rea **N**etwork

CARB
California **A**ir **R**esources **B**oard

CPLD
Complex **P**rogrammable **L**ogic **D**evice

CPU
Central **P**rocessing **U**nit

CR
Common **R**ail

D/A
Digital / **A**nalog

DPR
Dual **P**ort **R**AM

DSP
Digital **S**ignal **P**rocessor

EBI
Expansion **B**oard **I**nterface

ECU
Electronic **C**ontrol **U**nit (Steuergerät)

EMV
Elektro**m**agnetische **V**erträglichkeit

EOBD
European **O**n**b**oard **D**iagnostics

ESDL
Embedded **S**oftware **D**escription **L**anguage

ESP
Elektronisches **S**tabilitäts**p**rogramm

ETK
Emulator**t**ast**k**opf

FPGA
Field **P**rogrammable **G**ate **A**rray

FPU
Floating **P**oint **U**nit

HAL
Hardware **A**bstraction **L**ayer

HDL
Hardware **D**escription **L**anguage

HTML
Hyper**t**ext **M**arkup **L**anguage

HW
Hard**w**are

INCA
Mess-, Applikationssystem (**In**tegrated **C**alibration and **A**cquisition Systems)

IRQ
Interrupt **R**e**q**uest

JTAG
Joint **T**est **A**ction **G**roup

KW
Kurbel**w**elle

KWP
Key**w**ord **P**rotocol

LEV
Low **E**mission **V**ehicle

LEV II
Kalifornischer Abgasstandard für Fahrzeuge

LIN
Local **I**nterconnect **N**etwork (siehe auch LIN-Bus)

MIPS
Million **I**nstructions **per S**econd

MOST
Media **O**riented **S**ystems **T**ransport

NW
Nocken**w**elle

OBD II
On**b**oard **D**iagnostics (2. Stufe)

OS
Betriebssystem (**O**perating **S**ystem)

OSEK
Arbeitskreis **O**ffene **S**ysteme für die **E**lektronik im **K**raftfahrzeug

PCI
Peripheral **C**omponent **I**nterconnect (siehe auch PCI-Bus)

PDE
Pumpe-**D**üse-**E**inheit

PWM
Puls **W**eiten **M**odulation

QSPI
Queued **S**erial **P**eripheral **I**nterface

RAM
Random **A**ccess **M**emory (Schreib-/Lesespeicher)

ROM
Read-**O**nly **M**emory (Nur-Lesespeicher)

SG
Steuer**g**erät

SRAM
Static **R**andom **A**ccess **M**emory

SULEV
Super **U**ltra **L**ow **E**mission **V**ehicle

SW
Soft**w**are

Tbd.
To be defined

TIP
Target **I**ntegration **P**ackage

TIP Exp
TIP Experimental Target

TPU
Timing **P**rocessing **U**nit

UIS
Unit **Injector S**ystem

ULEV
Ultra **L**ow **E**mission **V**ehicle

UML
Unified **M**odeling **L**anguage

VHDL
VHSIC **H**ardware **D**escription **L**anguage

VHSIC
Very **H**igh **S**peed **I**ntegrated **C**ircuit

VME
Versa **M**odule **E**urocard (siehe auch VMEbus)

XML
E**x**tensible **M**arkup **L**anguage

ZEV
Zero **E**mission **V**ehicle

GLOSSAR

ASCET
Grafisches Entwicklungswerkzeug für die funktionale Entwicklung eingebetteter Softwaresysteme.

Adaptation Board
Aufgabenspezifisches Signalkonditionierungsboard, welches auf ein Mikro-controllerboard gesteckt werden kann.

Betriebssystem
Das Betriebssystem steuert den zeitlichen Ablauf der Ausführung/Aktivierung eines Softwaresystems. Das Betriebssystem leistet auch Dienste für die Kommunikation (Messages) und den Zugang zu reservierten Teilen der Hardware (Ressourcen). Das Betriebssystem des ASCET basiert auf dem Echtzeit-Betriebssystem ERCOS^{EK}.

Blockdiagramm
Ein Blockdiagramm ist eine grafische Beschreibung für eine Komponente, in der die verschiedenen Elemente, Operatoren und Eingänge (Argumente) sowie Ausgänge (Ergebniswerte) durch Richtungslinien miteinander verbunden sind. Ein Blockdiagramm besteht aus mehreren Diagrammen. Die Beschreibung als Blockdiagramm ist eine physikalische Beschreibung im Gegensatz zur Darstellung mittels C-Code.

BYPASS
Schnittstelle zum Austausch von Daten zwischen einem Steuergerät und einem Entwicklungssystem. Einige Berechnungen werden im Entwicklungssteuergerät durchgeführt.

Client
Anwendung welche Anfragen an einen Server sendet und die entsprechenden Antworten entgegennimmt.

Codegenerierung
Codegenerierung ist der erste Schritt bei der Umwandlung eines physischen Modells in einen ausführbaren Code. Das physikalische Modell wird in ANSI C-Code umgesetzt. Da der C-Code compilerabhängig (und damit targetabhängig) ist, wird für jedes Target ein spezifischer Code erzeugt.

DISTAB
Datenaustauschverfahren zwischen einem elektronischen Steuergerät und einem Entwicklungssystem.

ERCOS^{EK}
Echtzeit-Betriebssystem in Anlehnung an den OSEK Standard.

EURO IV
Europäische Abgasgesetzgebung für Pkw auf der Basis der Richtlinie 98/69/EG

Fullpass
Bypass, bei dem sämtliche Berechnungen auf das Entwicklungssystem ausgelagert sind.

Implementierung
Eine Implementierung beschreibt die Umsetzung der physikalischen Aufgabenstellung (des Modells) in ausführbarem Festkomma-Code. Eine Implementierung besteht aus einer (linearen) Umsetzungsformel und einem Begrenzungsintervall für die Modellwerte.

Intellectual Property
Ein Intellectual Property (IP) -Core ist eine vorgefertigte, mehrfach verwendbare Funktionseinheit, die in ein Design von rekonfigurierbarer Logik eingebunden werden kann.

Kennfeld
3-dimensionale Verstellgröße

Kenngröße
Oberbegriff für Kennfeld, Kennlinie und Kennwert (siehe auch Verstellgröße)

Kennlinie
2-dimensionale Verstellgröße

Kennwert
1-dimensionale Verstellgröße (Parameter)

Klasse
Eine Klasse ist ein Komponententyp in ASCET. Klassen in ASCET sind vergleichbar mit objektorientierten Klassen. Die Funktionalität einer Klasse wird durch Methoden beschrieben.

LIN-Bus
Der **L**ocal **I**nterconnect **N**etwork Bus ist ein einfaches serielles Kommunikationssystem, das für den Einsatz in Fahrzeugen entworfen wurde. Es ermöglicht die kostengünstige Verbindung räumlich verteilter elektronischer Systeme in Fahrzeugen und ergänzt damit die bestehenden komplexen Kommunikationssyteme.

Methode
Eine Methode ist Teil der Beschreibung der Funktionalität einer Klasse unter dem Aspekt der objektorientierten Programmierung. Eine Methode verfügt über Argumente und einen Rückgabewert.

Modul
Ein Modul ist ein Komponententyp in ASCET. Es beschreibt Prozesse, die vom Betriebssystem aktiviert werden können. Ein Modul kann nicht als Teilkomponente in anderen Komponenten eingesetzt werden.

MPC555
32 Bit Mikrocontroller der Firma Motorola®.

PB1640
Adaptation Board mit Signalkonditionierung für Diesel Einspritzung.

PCI-Bus
Der **P**eripheral **C**omponent **I**nterconnect Bus ist ein weitgehend CPU-unabhängiger, leistungsfähiger, paralleler Bus. Er wurde im Jahre 1991 von INTEL konzipiert um die Datenraten zu steigern und gleichzeitig die elektromagnetische Verträglichkeit (EMV) zu verbessern.

Projekt
Ein Projekt beschreibt ein gesamtes eingebettetes Softwaresystem. Es enthält Komponenten, welche die Funktionalität definieren, eine Betriebssystem- Spezifikation sowie einen Bindemechanismus, der die Kommunikation festlegt. Es enthält alle für die Applikation relevanten Verwaltungsinformationen (Adressen, Ablageschemata...) eines Programms. Einem Projekt und damit einem Programmstand können mehrere Arbeitsbasen zugeordnet werden. Damit ist das Projekt die interne Repräsentation einer ASAM-MCD-2MC Datei.

Prozess
Ein Prozess ist eine zeitgleich ausführbare Funktionalität, die durch das Betriebssystem aktiviert wird. Prozesse werden in Modulen spezifiziert und verfügen über keinerlei Argumente/Eingaben oder Ergebniswerte/Ausgaben.

Scheduling
Das Scheduling ist das Zuweisen von Prozessen zu Tasks und die Aktivierung von Tasks durch das Betriebssystem.

Sequencing
Festlegen der Berechnungsreihenfolge von Prozessen.

Server
Anwendung welche Anfragen von Clients entgegen nimmt und diese entsprechend beantwortet.

Target
Ein Target ist die Hardware, auf der ein Experiment läuft. Ein Target kann entweder ein experimentelles Target (PC, PowerPC) oder ein Mikrocontroller-Target sein.

Task
Eine Task ist eine geordnete Sammlung von Prozessen, die durch das Betriebssystem aktiviert werden können. Attribute einer Task sind ihre Betriebsarten, ihr Aktivierungs-trigger, ihre Priorität und der Modus ihres Schedulings. Bei Aktivierung werden die Prozesse der Task in der angegebenen Reihenfolge ausgeführt.

Trigger
Ein Trigger aktiviert die Ausführung einer Task (im Rahmen des Betriebssystems) oder eines Zustandsautomaten.

VMEbus
Der **V**ersa **M**odule **E**urocard Bus ist ein asynchroner, weitgehend CPU-unabhängiger, leistungsfähiger, paralleler Bus. Er wurde von den Firmen Motorola, Philips und Mostek konzipiert und 1981 freigegeben sowie dem IEEE und IEC zur Standardisierung vorgelegt.

Zustandsautomat
Verhaltensbeschreibung durch einen Zustandsgraphen, der aus durch Übergänge verbundenen Zuständen besteht.

LITERATURVERZEICHNIS

[AUTOSAR] AUTOSAR
Automotive Open System Architecture
http:\\www.autosar.org

[BBR+98] Bertram T., Bitzer R., Mayer R., Volkart A.
CARTRONIC
– An Open Architecture for Networking the Control Systems of an Automobile
SAE International Congress and Exposition
Detroit / USA, Februar 1998

[BHK00] Bürger K. G., Harms K., Kallenbach R.
Elektronische Systeme im Kraftfahrzeug
– Perspektiven für das nächste Jahrzehnt
Sonderausgabe ATZ und MTZ
Vieweg Verlag 2000

[Ber01] Bertram T.
Modellbildung und Simulation in der Entwicklung mechatronischer Systeme für das Kraftfahrzeug.
Zweites Fachforum für Mechatronik,
- funktions und kostenoptimierte Systemlösungen –
Regensburg 2001

[Ber02] Bertram T.
Eine Erfolgsgeschichte
– von der mechanischen zur mechatronischen Bremse –
47. Internationales Wissenschaftliches Kolloquium Maschinenbau und Nanotechnik,
TU Ilmenau 2002

[BO01] Bertram T., Opgen-Rhein P.
Modellbildung und Simulation mechatronischer Systeme,
- Virtueller Fahrversuch als Schlüsseltechnologie der Zukunft –
Atomotive Electronics September 2001

[Bos01] Robert BOSCH GmbH
Elektronische Dieselregelung EDC
1. Auflage, 2001
ISBN 3-7782-2035-7

[Bos02_1] Robert BOSCH GmbH
Dieselmotor-Management
3. Auflage,
Vieweg Verlag 2002
ISBN 3-528-13873-4

[Bos02_2] Robert BOSCH GmbH
Kraftfahrtechnisches Taschenbuch
24. Auflage,
Vieweg Verlag 2002
ISBN 3-528-13876-9

[BRK01] Beier Th., Richard A. H., Kullmer G.
Entwicklung eines intramedullären Implantats zur Knochenbruchheilung
4. VDI Mechatronik Tagung 2001
–Innovative Produktentwicklung
VDI Berichte 1631, September 2001

[Dou97] Doulos
The VHDL Golden Reference Guide
2. Auflage, 1997

[Dsp04] dSPACE GmbH
MicroAutoBox
Produktinformation
Februar 2004

[EDG01] Eppinger A., Dieterle W., Georg K.
Mechatronik
– Mit ganzheitlichem Ansatz zu erhöhter Funktionalität und Kundennutzen –
Automotive Electronics September 2001

[Ern99] Ernst R.
Automatisierter Entwurf eingebetteter Systeme
at Juli 1999

[Eta02_1] ETAS GmbH
ASCET-SD V4.2
Benutzerhandbuch
September 2002

[Eta02_2] ETAS GmbH
TIP Experimental Target V4.4
Benutzerhandbuch
September 2002

[Eta02_3] ETAS GmbH
ASCET-SD Target Integration Package V4.2: MPC5xx
Referenzhandbuch
März 2002

[Ets02] Etschberger K.
Controller **A**rea **N**etwork
3. Auflage, 2002
ISBN 3-446-21776-2

[Fla02] Flath M.
Methoden zur Konzipierung mechatronischer Produkte
Dissertation, Paderborn 2002
ISBN 3-935433-17-4

[Frü00] Frühauf F.
Aktive Fahrzeugfederung
Mechatronik – Mechanisch/Elektrische Antriebstechnik
VDI Berichte 1533, März 2000

[GEK01] Gausemeier J., Ebbesmeyer P., Kallmeyer F.
Produktinnovationen
Carl Hanser Verlag München 2001
ISBN 3-446-21631-6

[GHB98] Gerhardt J., Hönninger H., Bischof H.
A New Approach to Functional and Software Structure for Engine Management Systems
– BOSCH ME7
SAE International Congress and Exposition
Detroit / USA, Februar 1998

[GM03] Gausemeier J., Möhringer S.
Die neue Richtlinie VDI2206: Entwicklungsmethodik für mechatronische Systeme
5. VDI Mechatronik Tagung 2003
–Innovative Produktentwicklung
VDI Berichte 1753, Mai 2003

[GL00] Gausemeier J., Lückel J.
Entwicklungsumgebungen Mechatronik
HNI, Paderborn 2000
ISBN 3-931466-79-5

[IAV] IAV
Ingenieurgesellschaft Auto und Verkehr
http:\\www.iav.de

[IABG] IABG
Das V-Modell
http://www.v-modell.iabg.de

[IBM03] IBM Microelectronics Division
PowerPC 750GX Microprocessor
USA October 2003

[Ise02] Isermann R.
Mechatronische Systeme
2. Auflage
Springer Verlag 2002

[JKM+02] Jessen H., Kaiser L., Mencher B., Gerhardt J.
Schnittstellen und Strategien zur Momentenkoordination und Umsetzung in Motorsteuerungssystemen für Ottomotoren
AUTOREG 2002
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1672, April 2002

[Kar03] Karpur G.
Design and implementation of firmware routines for Motorola PowerPC as master processor for programming flash devices and PLDs from a remote PC over Ethernet and VMEbus
Diplomarbeit, Januar 2003

[Kas02] Kasper M.
Einführung in das zielsystemidentische Rapid Prototyping für elektronische Steuergeräte
AUTOREG 2002
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1672, April 2002

[Kas04] Kasper M.
Flexible Ansteuerung für magnetventilbasierte Komponenten
AUTOREG 2004
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1828, März 2004

[Kau02] Kaus E.
Fahrwerksmechatronik und deren Potential zur Steigerung von Komfort und Sicherheit
AUTOREG 2002
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1672, April 2002

[KFM+01] Kracke T., Fengler H. P., Müller P., Barsum N.
FI^2RE – Ein Entwicklungssteuergerät für flexible Einspritzung und Zündung
MTZ (62) Nr. 1
Vieweg Verlag 2001

[KM00] Köhler B., Mewes F.
Mechatronisches Pumpensystem durch intelligente, integrierte Antriebe
Mechatronik – Mechanisch/Elektrische Antriebstechnik
VDI Berichte 1533, März 2000

[Koc00] Koch W.
Eine interaktive Entwurfsplattform für mechatronische Systeme auf Basis von Komponentensoftware
Dissertation, Magdeburg 2000
VDI Verlag Düsseldorf 2000
ISBN 3-18-332020-7

[KP93] Kirch-Prinz U., Prinz P.
C für PCs
6. Auflage, 1993
ISBN 3-88322-215-1

[Küm99] Kümmel M. A.
Integration von Methoden und Werkzeugen zur Entwicklung von mechatronischen Systemen
Dissertation, Paderborn 2002
ISBN 3-935433-17-4

[KWG+00] Kaltenbach J., Werries H., Gazyakan Ü., Knödler H.
Entwicklungsprozeß für ein mechatronisches System am Beispiel Steer-by-wire,
Mechatronik – Mechanisch/Elektrische Antriebstechnik
VDI Berichte 1533, März 2000

[KZB+01] Kallenbach E., Zöppig V., Birli O., Feindt K., Ströhla T., Schaffert E., Schmidt J.
Integration mechatronischer Systeme
4. VDI Mechatronik Tagung 2001
–Innovative Produktentwicklung
VDI Berichte 1631, September 2001

[MBS+00] Müller-Glaser K., Burst A., Spitzer B., Kühl M.
Rapid Prototyping von eingebetteten elektronischen Systemen
Informationstechnik und Technische Informatik (it+ti) 2000
Volume 42, Issue 02

[Mer02] Mercer Management Consulting
Automobiltechnologie 2010
http://www.mercer.de

[MH02] Münch P., Hollstein J.
Stufenlose Getriebe und ihr Einsatz in der modernen Landtechnik
AUTOREG 2002
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1672, April 2002

[MI02] Müller N., Isermann R.
Zylinderdruck-basiertes Motormanagement beim Ottomotor
AUTOREG 2002
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1672, April 2002

[Mot00] Motorola GmbH
MPC555 User's Manual
October 2000

[OBH+02] Odenthal D., Bünte T., Heitzer H., Eicker Ch.
Übertragung des Lenkgefühls einer Servolenkung auf Steer-by-wire
AUTOREG 2002
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1672, April 2002

[OEM04] Otterbach R., Eckmann M., Mertens F.,
Rapid Control Prototyping
– neue Möglichkeiten und Werkzeuge
AUTOREG 2004
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1828, März 2004

[OSEK] OSEK
Open Systems and the corresponding interfaces for automotive electronics
http:\\www.osek-vdx.org

[PH02] Pfeiffer F., Haj-Frau, A.
Optimale Schaltregelung bei Stufenautomaten
AUTOREG 2002
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1672, April 2002

[PKS+04] Predelli O., Kracke T., Schmidt W., Meyer S.
FI^2RE – Neues Steuermodul für Piezoinjektoren
MTZ (65) Nr. 1
Vieweg Verlag 2004

[RG+03] Reuter J., Görg S.
Der IAV Engine Controller
Sonderausgabe ATZ und MTZ
Vieweg Verlag 2003

[Ric] Ricardo Consulting Engineers Ltd.
Control & Electronics
http:\\www.ricardo.com

[SBR03] Seibertz A., Busch R., Requejo J.
ECP – Das fehlende Glied in der Entwicklungskette dargestellt am Anwendungsbeispiel IVDC
5. VDI Mechatronik Tagung 2003
–Innovative Produktentwicklung
VDI Berichte 1753, Mai 2003

[Sch96] Schäffel Ch.
Untersuchungen zur Gestaltung integrierter Mehrkoordinatenantriebe
Dissertation, Ilmenau 1996

[Sch00] Schwankl L.
Konzeptionelle Entwicklung innovativer Bremssysteme
Mechatronik – Mechanisch/Elektrische Antriebstechnik
VDI Berichte 1533, März 2000

[SPZ+01] Schwankl L., Pichler B., Ziegler S., Bernecker F., Hanschke M.
Interdisziplinäre Entwicklung komplexer mechatronischer Produkte – am Beispiel medizintechnischer Anwendungen
4. VDI Mechatronik Tagung 2001
–Innovative Produktentwicklung
VDI Berichte 1631, September 2001

[SR02] Schwarz R., Rieth P.
Global Chassis Control
– Systemvernetzung im Fahrzeug
AUTOREG 2002
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1672, April 2002

[SZ03] Schäufele J., Zurawka T.
Automotive Software Engineering
ATZ-MTZ Fachbuch
1. Auflage, 2003
Vieweg Verlag 2002

[TS91] Tietze U., Schenk Ch.
Halbleiter- Schaltungstechnik
9. Auflage, 1991
ISBN 3-540-19475-4

[VDI03] VDI2206: 2003-03 Entwicklungsmethodik für mechatronische Systeme
(Entwurf)
Beuth Verlag, Berlin
März 2003

[WCF+03] Winter A., Castellanos E., Förch U., Kasper M.
Ein universelles Entwicklungssteuergerät für die Funktionsentwicklung und ein zielsystemidentisches Rapid Prototyping System
TZ Si 2003
Februar 2003

[WFK04] Winter A., Förch U., Kasper M.
Rapid Prototyping am Beispiel eines Energiemanagementprojektes
AUTOREG 2004
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1828, März 2004

[Wen04] Wenzel T.
ETK – Interface for ECU Calibration
ETAS Technical Document
April 2004

[Xil01] XILINX GmbH
Virtex Field Programmable
Gate Arrays
April 2001

[Xil02] XILINX GmbH
XCR3256XL
256 Macrocell CPLD
November 2002

[Zim02] Zimmer M. G.
Steer-by-wire
– Eine Herausforderung
AUTOREG 2002
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1672, April 2002

[ZL02] Zöllner S., Landsmann G.
Brennraumdruckgeführtes Motormanagement beim Ottomotor
– Funktionen und Potenziale
AUTOREG 2002
–Steuerung und Regelung von Fahrzeugen und Motoren
VDI Berichte 1672, April 2002

INDEX